solidThinking™

User Manual

Vol. 3 Rendering

solidThinking™

Release 7.6

User Manual

Rendering

© 1993-2007 solidThinking Inc.
Part N. 80-25041-10301

Copyright © 1993-2008 solidThinking, Inc. All rights reserved.

solidThinking™ and *renderThinking*™ are trademarks of solidThinking, Inc. All other trademarks or servicemarks are the property of their respective owners.

This documentation and the software program are copyright solidThinking, Inc. and your rights are subject to the limitations and restrictions imposed by the copyright laws. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, or disclosed to a third party, without the prior written permission of solidThinking, Inc.

solidThinking, Inc. reserves the right to make changes in specifications at any time and without notice. Parts of this documentation are copyright LightWork Design Ltd. 1990-2008 ©. solidThinking Inc. provides this publication “as is” without warranty of any kind, either expressed or implied.

Cover: The Handle, a XOX Audio Tools guitar invented and modeled in solidThinking. Design by Peter Solomon, 3D modeling and engineering by Emilio Cassani.

Table of Contents

Rendering

Section 5
Rendering in solidThinking 1

5.1 Shaders 7
Shader Classes 8
5.2 Materials 12
Creating Materials 13
Assigning Materials 17
Merging Materials 18
The Materials LIbrary 19
Editing Materials 21
Materials Components 22
5.3 Textures 103
The Texture Coordinate System 105
Texture Space Shaders Description 106
Texture Placement 116
Interactive Texturing 126
5.4 Lights 133
Light Source types 134
New Light 136
Assigning a Light Shader 139
Saving Light Settings 141
Editing Light Settings 142

Shadows 143
Controlling Shadow Production for Each Object 145
Light Shaders 146
5.5 The Camera 163
New Camera 164
5.6 Backgrounds and Foregrounds 167
Background Shaders 169
Foreground Shaders 180
5.7 Sketch Styles 193
5.8 Radiosity 207
How the Radiosity Technique Works 209
Computing Radiosity Values 210
Defining Lights for Radiosity 212
Defining Materials 217
5.9 Final Gather 218
5.10 Feature Following Anti Aliasing 223
5.11 Depth of Field 225
5.12 Lens Flares 227
5.12 Image Output Formats 229
5.13 Rendering the scene 235
The Render Options Panel 239
The Renderings Browser 241
Interreflections 242
Creating an Image Step by Step 243
Rendering Hints 246

Glossary ***249***
Shortcuts for Windows ***257***
Shortcuts for Macintosh ***263***

Section 5

Rendering in solidThinking

5 Rendering in solidThinking

This section describes how to render realistic images using the advanced shading techniques available in the ***renderThinking*** rendering module. ***solidThinking*** uses ***renderThinking*** as the main rendering engine but it is open to support other renderers, including RenderMan compliant renderers.

Included are details connected to shaded rendering: from scan-line rendering, use of shaders and shader libraries to texturing, materials, environment mapping and ray tracing.

Shading can produce images that are much easier to understand compared to line-drawn images, which are familiar to engineers but not to managers and other non-technical people. The shading of geometry according to its orientation to the light sources in a scene provides the viewer with important visual clues to the arrangement of the objects in three-dimensional space, even though they are rendered onto a two-dimensional image.

The simplest form of shading provided in ***renderThinking*** is Flat shading, where the orientation of a surface to the light source(s) is established, and the whole surface then shaded lighter if the surface is facing toward light sources, darker if facing away. Even though this method leads to images that appear to contain solid, three-dimensional objects, geometry is ultimately represented as flat polygons; therefore this simplistic method of shading results in images in which the facets of the geometry are apparent.

Two more sophisticated shading methods are Gouraud shading and Phong shading. The Gouraud method 'smoothes' the shading using a linear interpolation between the vertices, providing a good approximation of smooth matte surfaces. The Phong method, on the other hand, mimics the effects of highlights by interpolating the normal vector of the surface between the vertices, and then using this normal as the starting point for a new shading calculation at each pixel. This latter method gives a good approximation of smooth surfaces that can exhibit specular highlights.

Antialiasing techniques are also available to reduce the jagged appearance of silhouettes and texture patterns, by area-sampling for each pixel, so as to deduce an average color for each pixel which is representative of all of the geometry and patterning contained within the area of the pixel.

It is also possible attaching a texture to a surface during this rendering process, which is often a good technique for adding realism to a scene. This can be done in one of two ways: 1) an image can be scanned into an image file in some way, and then applied to a surface with a given orientation and (image texture mapping); 2) a texture can be defined in a mathematical way, so that a procedure can be used to establish what the correct color of a given point on the surface would be (procedural texture).

Mirror-like reflections are another way of making images look much more realistic. These can also be simulated in two ways.

A first technique is similar to texture mapping and is called environment mapping: a virtual image created by the software is used to simulate the environment surrounding the geometry.

A second technique for handling reflectivity of surfaces is to use a technique known as ray tracing. This involves tracing the path of light rays from the view point as they bounce off the surface and travel toward other objects in the scene in order to establish the origin of each ray. It is possible to render a whole scene using only ray tracing, rather than any scan-line techniques: the disadvantage of such an approach, however, is that the technique is computationally intensive, and often slower than scan-line algorithms. ***renderThinking*** allows you to set the number of interreflections computed when you use the raytrace method. The higher the number, the slower the computation (*see, for more details, 5.13 Interreflections*). In ***renderThinking*** it is also possible to mix scan-line and ray-tracing techniques, using scan-line algorithms for visibility calculations and ray-tracing merely to handle reflections, refractions or shadows when required.

Any number of light sources can be positioned within the three-dimensional scene along with the objects of interest, so as to produce shading and shadows which are representative of particular lighting conditions. These light sources can be specified as having particular intensities and colors, such as: point lights, which radiate light equally in all directions; distant lights for modeling sunlight; spot lights which produce a directional cone of light.

It is also possible to represent transparent materials, but which alter the color of objects seen through them, and may reflect a portion of the light incident on them, thus giving them a shiny appearance.

Patterns can be attached to objects which are matte or shiny, small undulations to the surface can be applied to any object, regardless of its color, reflectivity or transparency, and so on.

A mechanism is provided whereby the overall appearance of an object is specified in terms of a 'material' definition.

A material definition is made up of several components, specifying such matters as the reflectivity, transparency, color, texture, and so on. These material definitions can be associated with any number of objects, and the individual shader components can be used in any number of material definitions, so that the appearance of all the geometry in a whole scene can quickly be built up from a library of shader components.

Radiosity based light simulation is also available in ***renderThinking***, and it is described in a separate section. Radiosity gives a stunning level of realism due to its accurate simulation of diffuse lighting conditions.

renderThinking also provides a mechanism for performing post-processing operations on images once rendered. This is achieved by rendering into a screen buffer, upon which post-processing is done before outputting in the normal way. Various effects can be achieved using image post-processing, such as lens flare, depth of field, and contrast adjustment.

The Sketch rendering method is used to create stylised presentations. In contrast to photorealistic rendering, the aim of Sketch rendering is not to use the computer to produce an image which looks as close as possible to a photograph, but, instead, to use the computer to produce an image which is more like a pencil sketch, ink drawing, or painting.

The new release introduces Final Gather: a technique that can be used with or without Radiosity and which will calculate the effects of diffuse secondary illumination (i.e. bounced light) in a scene. It involves computing the incoming illumination at various points in a scene and storing this data in a cache. Information stored in the cache is then used to determine how illumination from other parts of the scene contribute to the illumination at the point being shaded. Thanks to Final Gather extremely accurate and well-lit images can be produced very quickly.

5.1 Shaders

renderThinking uses shaders for the purpose of evaluating specific shading calculations and, consequently, defining the appearance of the objects.

A shader can be defined as a C-code function that is provided with some data which defines its operating environment in the form of input variables. Based upon these variables, a result is calculated and communicated back to the renderer by means of output variables.

Simply stated, it is a collection of information describing all the surface peculiarities of an object.

A shader can be controlled precisely by means of arguments that define its specific function. Shaders can have zero, one or more arguments and these may each be of a simple or composite data type, including integers, floating point values, colors, vectors and enumerations.

A shader can be seen as a 'black box', which is expected to perform some computations and provide its results to the rendering subsystem unseen by the user, via a number of global variables.

Shader Classes

There are different classes of shaders, each usable in a particular circumstance and for a specific purpose. Here is a brief description of these classes:

Color Source Shader

A color source shader is used to define the color of a surface at any point in space. It may be as simple as a plain color (which specifies all parts of the surface to have a uniform color), or it may define complex surface patterns such as marble or wood.

Every surface must have a color source shader associated with it, and for a given surface its color source shader will only be called for those pixels (if any) at which the surface is known to be visible.

Reflectance Model Shader

The behavior of a surface in the presence of light is represented by a reflectance model shader that defines how much light is reflected by the surface towards the viewer. Thus, this result depends upon both the material properties of the surface and the lighting environment that illuminates the surface.

Unlike the color source shaders that specify the pure color of a surface and are therefore independent of any light sources, reflectance model shaders must account for each light source in calculating their results. Shaders of this type may be thought of as defining a surface's "finish" and are used to model reflectance properties such as matte, metal and plastic.

Transparency Source Shader

Similarly to the color source shader that specifies the color of a surface, a transparency source is used to define how transparent or opaque a surface is, and thus how much light is able to pass through it. Internally, the rendering engine records transparencies as color values - which may be thought of as representing filters - and this enables complex surface effects to be created.

Transparency source shaders range from a simple plain uniform transparency to more complex regular or irregular eroded patterns that would be difficult to represent using modeling techniques.

Displacement Shader

Small surface perturbations can be supported by means of displacement shaders. Typically, a displacement shader will give an irregular or indented appearance to an otherwise smooth surface by modifying the surface normal vector that is used in subsequent shading calculations.

Displacement shaders are used to represent features that would be difficult, impossible or inefficient if conventional modeling techniques were used. For example, rough metal castings and the regular indentations produced by pressed sheet metal can be simulated.

Texture Space Shader

Texture space shaders are a little different from other classes of shader since the contribution that they make to the final intensity of pixels is less obvious. Their effect. upon perceived pixel intensities, is indirect rather than direct, since they serve to modify the environment within which color source, reflectance model, transparency source and displacement shaders operate. These latter four shader classes may opt to perform their calculations based upon a two-dimensional coordinate system known as texture space, and it is this coordinate system which is defined by a texture space shader.

Texture space shaders are used to 'wrap' the effects of shaders of other classes around surfaces in predefined ways. Planar, cylindrical and spherical mappings can all be supported.

Light Source Shader

Light sources are supported internally using a similar mechanism to the other shader classes. A light source shader defines which and how much light falls onto a surface.

Examples of light source types are ambient lights (which illuminate all surfaces equally regardless of their orientation), distant lights (which radiate light in a single direction), point lights (which radiate light in all directions), and spot lights (which cast a cone of light from a specified point and in a specific direction).

Emission Shader

Emission shaders are defined by geometry and material.
Examples of emission shaders are Area light sources or Portal lights, i.e. parallel light from a distant source, coming through a window.

Background Shader

For those pixels or parts of pixels that are not covered by any objects, intensity is specified by means of a background shader. A background shader may be as simple as one providing a single, uniform color, as well as one producing graduated and complex patterns.

A background shader can also be used to perform simple compositing functions by overlaying the computed image on top of a previously computed image or a scanned photograph.

Foreground Shader

The foreground shader class enables post-processing operations to be applied to the image during the shading pipeline.

Foreground shaders may be used to simulate atmospheric effects such as fog, or depth cueing. They may also be used in combination with background shaders to perform compositing operations.

Post-processing Shader

Post-processing shaders work on image data which has already been “rendered” into the screen buffer. The data in this buffer can then be manipulated to produce certain effects, such as lens flare and depth of field.

5.2 Materials

The information, which defines how an object will appear in shaded rendering methods, is specified by its material. A material may be assigned to any number of objects simultaneously.

With shaded rendering methods, the appearance of an object is basically derived from four independent shaders:

- color source,
- reflectance model,
- transparency source,
- displacement function.

Shaders that make up a material are known as its *components*.

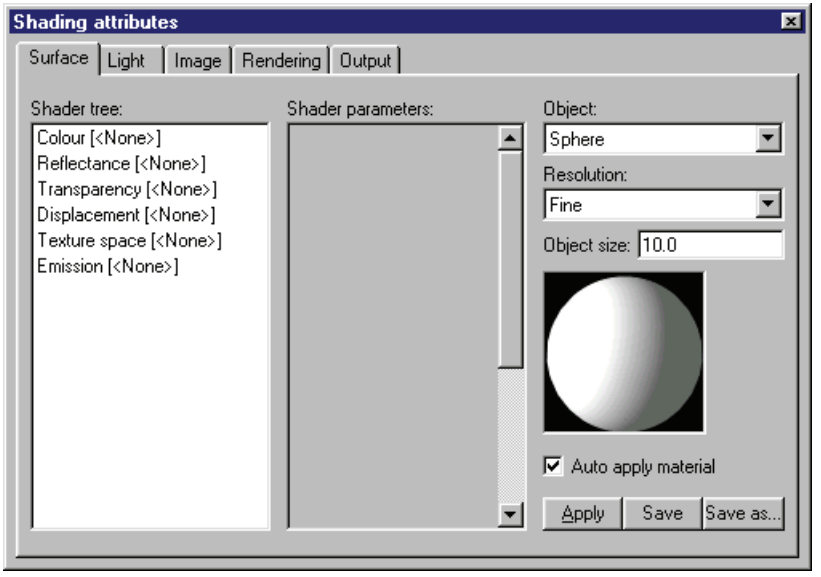
This model provides a highly flexible and versatile environment for the specification of material properties. Each component makes an independent contribution to a material's appearance, and consequently, each performs a specific function. Combining simple shading functions into each material can represent complex properties.

Creating Materials

Creation of materials requires you to first open the **Shading** panel by using either the **Managers ▸ Shading** command or the Ctrl + 3 shortcut.

Once opened, the **Shading** panel displays five tabs (Surface, Light, Image, Rendering, and Output) on the upper left corner.

To create materials applicable to objects, select the **Surface** tab:



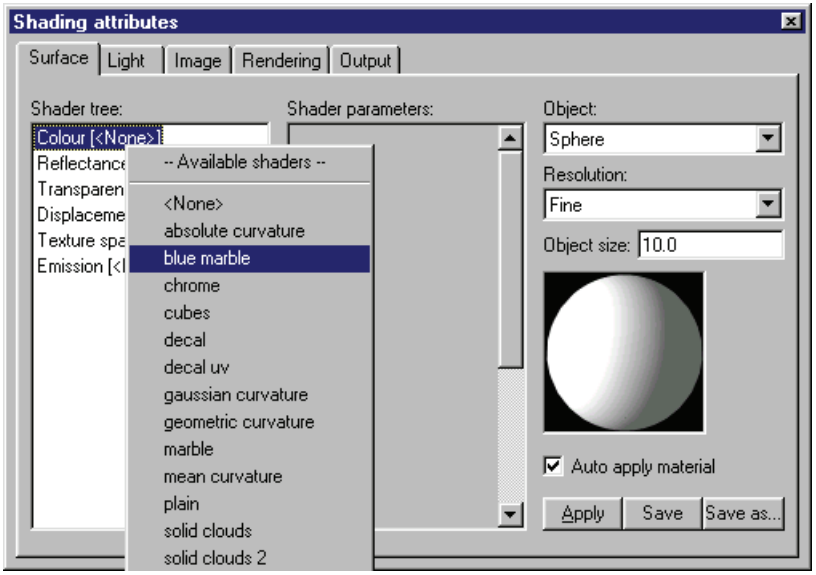
The **Shader tree** column on the left lists the shader types:

- **Color**
- **Reflectance**
- **Transparency**
- **Displacement**
- **Texture space**
- **Emission**

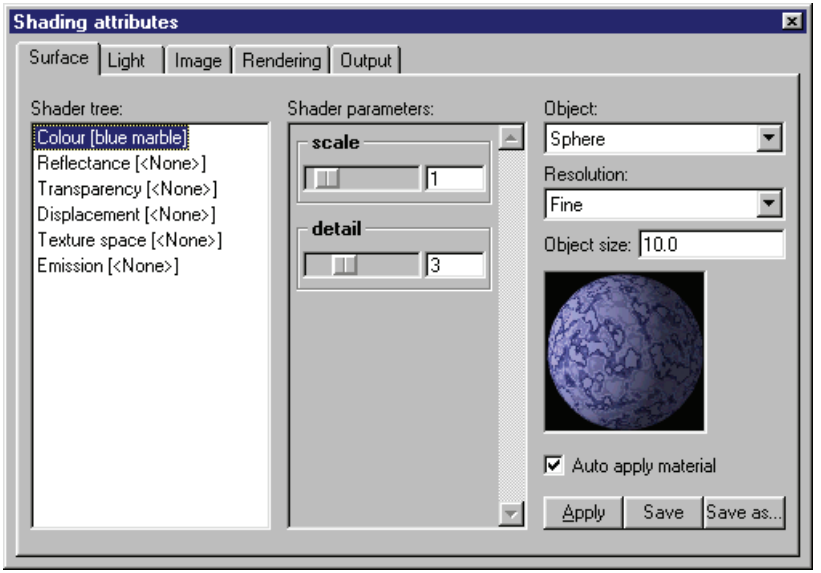
The name of the shader assigned is displayed between [square brackets]. <None> indicates that no shader of that class is currently assigned.

To assign a shader, just press the right mouse button when you are over a shader class: a pop-up menu will be displayed allowing you to scroll through the list of available shader of that class.

In our example, we press the right mouse button over the **Color** class and choose “blue marble”.



When you are over the shader you want to use, left click: the name of the chosen shader will be displayed, instead of <None>, in square brackets. In our example, you can read “blue marble”.



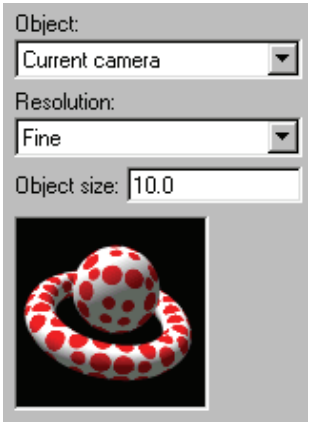
In the middle of the panel, the **Parameters** column lists the shader arguments by means of which you can customize the shader. Depending on the number of arguments a shader has, the list can be scrollable or not.

As we have said before, Color, Reflectance, Transparency, and Displacement shaders are the four possible components of a material. Obviously, you do not always need to use all of them to define a material.

The Texture space shader class allows you to define the texture space when one of the four components above includes a wrapped texture. Texture mapping will be examined later in a specific paragraph.

In the Emission shader class you only find Area light. This can be used exclusively with Radiosity. Please refer to the Radiosity paragraph for details on how to use this powerful shader.

On the top right of the panel, the **Object** pop-up menu lets you choose the preview object shown in the small preview window.



You can choose among:

- **None**,
- **Current camera**,
- **Current object**,
- **Sphere** (default),
- **Cylinder**,
- **Cube**
- **XY plane**,
- **XZ plane**,
- **YZ plane**.

***Note**: When you choose the Current camera option, **solidThinking** updates in real time the Preview window at any modification of what the current camera sees. You should remember that this could slow down operations. Use it when needed.*

Similarly, you can set the **Resolution**. Options are:

- **Fine**,
- **Medium**,
- **Coarse**.

Use **Object size** to change size of the preview object.

Assigning Materials

By default, a material defined in the Shading panel is automatically applied to any previously selected object (objects). In the same way any modification of a shader parameter is immediately applied. If you do not want this to happen, deselect the **Auto apply material** option. You will then have to press **Apply** whenever you want to assign the current material definition to the selected object.

***Note**: The **Auto apply material** option could slow down operations if you need to continually adjust material settings in a complex scene. In this case, you could decide to deselect this option.*

Two methods are available for assigning materials. The first one, which can be used when a material has to be created, includes the following steps:

1) Select the object (objects) to which you want to assign a material.
2) Create the desired material in the Shading panel.
3) Apply it as above.

Steps to assign a material you previously stored in the Materials library, are:

1) Select the object (objects) that you want to assign a material to.
2) Choose the material from the Materials library and click its icon: it will automatically be applied to the selected object (objects).

***Note**: When you assign environment, background, or foreground materials from the Materials library by clicking its related icon, no objects must be selected, as these materials are assigned to the scene.*

Merging Materials

When you click a material icon from the Materials library,you assign that material to any selected object (objects). Any material that you have previously assigned to the selected object (objects) will automatically be replaced.

You can, otherwise, obtain a partial merging of two materials if you click a material icon while holding the Ctrl key down: any shader of the clicked material that belongs to a class not included in the old material will be added as a new component. This way, you will obtain a third material with some components from the old material and others from the clicked one.

An example will probably help you understand this concept: A sphere has a material with one component only: a “plain” color shader.

After selecting the sphere, we click another material from the Materials library while holding the Ctrl key down. If this second material has two components, a “marble” color shader and a “rough” displacement shader for instance, the resulting material will have the original “plain” color shader but enriched by the “rough” displacement shader.

The Materials Library

solidThinking allows you to save any material you create and store it in the extensible **Materials library** so that it can be quickly retrieved for any successive use.

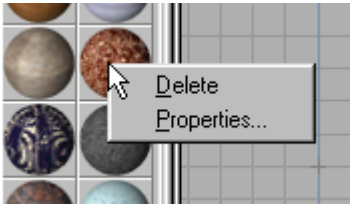
To save any new material, press the **Save** button on the bottom right of the Shading panel.

A small **Save as** panel is displayed. Choose the **Group** where you want to store the new material by scrolling the pop-up list. If you want to create a new group, simply type a different name not included in the pop-up list. Then, type the **Material** name and click **OK** to confirm.

You will find the new material in the Materials library under the group you chose.

***Note**: when naming groups and materials, follow some rules to ease retrieving operations and to avoid chaos in the Materials library.*

If you quickly right click a library material icon, the program displays a pop-up menu that shows you two options: **Delete** and **Properties**.



Choose Delete if you want to remove a material from the library, Properties if you want to rename it or move it to a different group.

Editing Materials

Any or all of the four shader components that make up a material may be modified at any time by changing argument values.

After editing a material assigned to a selected object, you have two options:
a) To overwrite the original material and save changes under the same name, simply press the **Save** button.
b) To retain the original material and create a new one under a new name, press the **Save as…** button.

The **Save as Material** panel is displayed. Choose the **Group** where you want to store the new material by scrolling the pop-up list, then type the **Material** name and click **OK** to confirm.

Materials Components

Let's now examine in detail the shaders of the four classes (color, reflectance, displacement, and transparency) that make up a material.

For each shader we are providing synopsis, arguments, related default values, and description.

You can become familiar with each shader by modifying arguments and verifying how this affects appearance.

Arguments provide the means to customize shaders, and to make them more powerful. For example, a red distant light and a white distant light are derived from the same shader type, but their specific properties are set by modifying the shaders' arguments (via a "color" argument, for example).

Each argument has a type and a value. A shader argument may be any one of the following types: integer, floating point, color, vector, string, boolean, enumeration, or pointer. A shader may have zero, one or more arguments.

"absolute curvature"

Class **Color**

Synopsis False color shader that shows the absolute 'curvature' of a surface.

Description The absolute curvature of the surface is represented by a false color 'rainbow'. This is done by converting the curvature into an HLS color value. In order to provide different colors for the minimal and maximal values, the range of the curvature changes is mapped onto a range hue of only [0, 300], rather than the full [0, 360]. The following table illustrates the color changes:

hue value	*color name*
0	Red
60	Yellow
120	Green
180	Cyan
240	Blue
300	Magenta

The parameter "**min curvature**" specifies the degree of curvature to be mapped onto hue 0, while "**max curvature**" specifies the degree to be mapped onto hue 300. The other colors are assigned using a linear interpolation.

"birch"

Class **Color**

Synopsis A version of the general "wood" advanced wood color shader with pre-defined values appropriate to a birch-like appearance.

Description This shader is simply a copy of the general-purpose "wood" shader where all the argument values have been pre-chosen to produce the appearance of birch. The arguments may of course be fine-tuned to suit particular jobs.

The orientation of the tree from which the wood is taken is specified by a point which lies on the axis in argument "**trunk centre**" and a direction vector pointing along the axis of the tree in argument "**trunk direction**".

An overall scaling factor is supplied in argument "**pattern scale**", which determines the radius difference of two adjacent rings. 0.01 means that there are 100 rings per unit length along the trunk radius. Choose a value appropriate to the units in which your geometry is modeled. The default value is appropriate for modeling in meters.

"blue marble"

Class **Color**

Synopsis A blue marble pattern.

Description A solid texture giving a blue marble appearance.

The detail or complexity of the texture may be controlled by means of the "**detail**" argument. A value of 1 will give a simple outline of the marble pattern, while a larger value such as 5 or 6 will give fine detail in the texture.

An overall scaling factor is provided in argument "**scale**"; increasing this value will make the marble texture appear larger.

"cherry"

Class **Color**

Synopsis A version of the general "wood" advanced wood color shader with pre-defined values appropriate to a cherrywood-like appearance.

Description This shader is simply a copy of the general-purpose "wood" shader where all the argument values have been pre-chosen to produce the appearance of cherrywood. The arguments may of course be fine-tuned to suit particular jobs.

The orientation of the tree from which the wood is taken is specified by a point which lies on the axis in argument "**trunk centre**" and a direction vector pointing along the axis of the tree in argument "**trunk direction**".

An overall scaling factor is supplied in argument "**pattern scale**", which determines the radius difference of two adjacent rings. 0.01 means that there are 100 rings per unit length along the trunk radius. Choose a value appropriate to the units in which your geometry is modeled. The default value is appropriate for modeling in meters.

"chrome"

Class **Color**

Synopsis Chrome-like reflections.

Description A color source providing simple chrome-like reflections.

The source has a base color, specified by argument "**base color**", which is mixed with bands of colors based on the orientation of the surface relative to a direction specified by argument "**vector**".

A "**mix**" argument is used to specify the ratio of mixing of base color with the reflection colors. This value should lie within the range 0 to 1, where a value of 1 indicates all base color and no reflection colors.

"cubes"

Class **Color**

Synopsis A three-dimensional lattice of cubes with alternating colors.

Description A solid texture corresponding to a three-dimensional lattice of cubes with alternating colors.

The cube colors may be specified by arguments "**odd color**" and "**even color**". The lattice is aligned with the axis of the local coordinate system. The size of each cube measured in the local coordinate system is set by argument "**scale**".

"decal"

Class **Color**

Synopsis A color shader providing support for shade trees.

Description A color shader that provides the entry point to support color shade trees.

This shader switches between two color-input channels. The current 'layer' is presented by the "**decal color**" shader, and the layer beneath by the "**base color**" shader. The switching between the two is accomplished by the alpha shader global set by the "decal transparency" shader.

Both the "decal color" and "**decal transparency**" shaders are executed using the texture space supplied as "**decal texture space**".

Note that "decal color" and "base color" may themselves be instances of the "decal" color shader. This enables hierarchies of color shaders to be assembled into shade trees.

"decal uv"

Class **Color**

Synopsis A color shader specific for labeling.

Description A color shader that allows you to label any image over a surface.

The name of the file containing the image data is provided as a string to parameter "**file name**".

Arguments «x origin» and «y origin» let you specify the origin in the uv space.

Independent scaling factors are provided for the u- and v-coordinate directions in arguments "**u scale**" and "**v scale**" respectively. This shader also allows you to change the "**angle** of the image.

The parameter "**inside coverage**" determines the ratio of area covered by the label to the area not covered: a value of zero means the label is completely transparent. "**outside coverage**" determines the ratio of area covered by the background to the area not covered: a value of zero means the background is completely transparent.

The sharpness of the label edges is controlled by "**u fuzz**" and "**v fuzz**". A value of zero gives very sharply defined edges; a value of one gives very fuzzy edges.

"draft angle evaluation"

Class **Color**

Synopsis Shader for evaluation mold shapes.

Description Surface evaluation for mold design. The "**pull direction**" parameter is direction in which the cast object is supposed to be pulled out of the mold. It will not be possible to pull it out if part of mold "overhangs" the object i.e. angle between pull direction and surface normal is smaller than 90 deg. If there is a part of surface that has such property then it will be marked with "**overhang color**". Next it will be impossible to pull the object out if tangent to the surface is parallel or almost parallel to the pull direction. They must differ by at least "**draft angle**" (usually 1 deg). So the parts of surface with normals that form angles between 90 deg and 90 deg + "draft angle" will be marked with "**fail color**".

The parts of surface with normals that fall into range between (90 deg + "draft angle") and (90 deg + "draft angle" + "tolerance angle") can be pulled out, although with some difficulty. So they are marked with "**warning color**".

Finally if the surface normal forms any angle between (90 deg + "draft angle" + "tolerance angle") and 180 deg with "pull direction" everything is nice and those areas are marked with "**pass color**". Each of the angles must fall into the range (0, 90) deg and also their sum must fall into this range. Otherwise the default angle values are used. The length of the pull direction vector must be greater than 0.0001. Otherwise the default pull direction is used.

Setting 0.0 for "draft angle" results in "fail color" zone disappearing. Setting 0.0 for "**tolerance angle**" results in "warning color" zone disappearing.

Setting both angles to 0.0 results in having displayed "pass color" for normal directions opposite to "pull direction" and "overhang color" for normal directions consistent with "pull direction".

"gaussian curvature"

Class **Color**

Synopsis False color shader that shows the 'gaussian curvature' of a surface.

Description The gaussian curvature of the surface is represented by a false color 'rainbow'. This is done by converting the curvature into an HLS color value. In order to provide different colors for the minimal and maximal values the range of curvature changes is mapped onto a range hue of only [0, 300], rather than the full [0, 360]. The following table illustrates the color changes:

hue value	*color name*
0	Red
60	Yellow
120	Green
180	Cyan
240	Blue
300	Magenta

The parameter "**min curvature**" specifies the degree of curvature to be mapped onto hue 0, whereas "**max curvature**" specifies the degree to be mapped onto hue 300. The other colors are assigned using a linear interpolation.

"geometric curvature"

Class **Color**

Synopsis False color shader which shows the selected type of surface curvature (absolute, mean or gaussian) by false color encoding.

Description This shader provides a more general method for false color encoding than the "absolute curvature", "gaussian curvature" and "mean curvature" shaders, which are simpler to use but less flexible.

The type of curvature to be revealed by the false color scheme is specified by parameter "**curvature type**", which should take one of the following values: "gaussian", "mean" or "absolute".

The color encoding is specified by using 6 parameters which represent the minimum curvature value (and its color), the maximum curvature value (and its color), the number of divisions of color to be displayed between these two extremes, and a matching array of colors to be used for the divisions (if this array is not supplied then a simple interpolation between the two extreme colors is used).

"granite"

Class **Color**

Synopsis A solid texture giving a granite like pattern.

Description A color shader that imitates the appearance of granite. Granite looks the way it does because it is composed of several different sorts of minerals which form grains of various sizes. Different types of granite differ in the relative number and sizes of the different grains of mineral. The argument "**type**" is used to specify the following common types of granite:

type	*description*
sierra	Small variations in grain size, minerals distributed uniformly.
bianco	Bigger variations in grain size, minerals distributed uniformly.
diamond	Domination of coarse grains, with small addition of fine-grained minerals.
azalea	Fine-grained with small addition of large-grained mineral.
auburn	Very large grains of the dominating mineral, the remaining minerals being distributed uniformly in small grains.

In addition to these standard pre-defined types of granite, it is possible to create you own unique types of granite using the other shader arguments described below.

The granite pattern is made up of four seperate minerals, the average grain size of each mineral being different.

The minerals are numbered in order of decreasing grain size, and the color of each can be specified, so argument "**mineral 1 color**" specifies the color of the mineral which causes the largest grains, and "**mineral 4 color**" specifies the color of the mineral which causes the smallest type of grains.

The overall granite texture can be scaled using the "**scale**" argument (the smaller this value is, the smaller all of the granite grains will appear). The curvature of the borders or edges of the grains can be controlled using the arguments "**fragment softness**" (which affects strength of random deformation of grains) and "**fragment detail**" (which determines scale of this deformation). A value of 1.0 for "**fragment detail**" defines smooth deformations, larger values increase frequency and deformations become denser and more rough.

Every mineral grain can be subdivided into smaller fragments, within which colors vary slightly from the color determined for the mineral to which a grain belongs. The argument "**fragment size**" describes the average size of such fragments, relative to the size of the largest grains. Grains whose size is smaller or equal to this argument value are not divided into smaller fragments. Note that "fragment size" must be lower or equal to 1.0, and greater than 0.0. A value of 1.0 for this argument means there will be no subdivision. The color of the fragments constituting each mineral grain, can also vary from the base mineral color. This is controlled by the "**color variation**" argument, which can vary from 0.0 (no differences) to 1.0 (maximum differences).

The effect of `cracks' on the borders between fragments can be introduced by setting the argument "**cracks**" to a value greater than 0.0. A value of 1.0 produces brightest cracks. The argument "**color noise**" defines the intensity of the randomly distributed noise that will be applied to texture.

The noise is superimposed uniformly on all mineral grains appearing in the pattern. The density of the noise is controlled by the argument "**color noise scale**", which should be defined relative to "scale". This means that setting this argument to 1.0 causes the average size of the random spots added to the pattern to be comparable to the size of the largest grains.

Finally, the argument "**random seed**" allows you to change the random distribution of grains. Different values will produce various patterns.

"maple"

Class **Color**

Synopsis A version of the general "wood" advanced wood color shader with pre-defined values appropriate to a maple-like appearance.

Description This shader is simply a copy of the general-purpose "wood" shader where all the argument values have been pre-chosen to produce the appearance of maple. The arguments may of course be fine-tuned to suit particular jobs.

The orientation of the tree from which the wood is taken is specified by a point which lies on the axis in argument "trunk centre" and a direction vector pointing along the axis of the tree in argument "**trunk direction**".

An overall scaling factor is supplied in argument "**pattern scale**", which determines the radius difference of two adjacent rings. 0.01 means that there are 100 rings per unit length along the trunk radius. Choose a value appropriate to the units in which your geometry is modeled. The default value is appropriate for modeling in meters.

"marble"

Class **Color**

Synopsis A marble pattern.

Description A solid texture giving a veined marble appearance.

The color of the ground of the marble is set through the "**ground color**" parameter, and the base color of the veins is set through "**vein color**".

The contrast of the color bands within the veins may be varied by the "**vein contrast**" parameter, for which larger values produce a greater contrast of colors.

Three parameters control the characteristics of the turbulent marble pattern: The level of detail in the bands of the marble is set through the "**detail**" parameter. Increasing this value has the effect of increasing the complexity of, and detail in, the pattern.

The marble has a crystalline structure, and the amplitude of the marble crystal grain is set with "**grain**"; larger values accentuate the granularity.

The size of the crystal grain is set through the "**grain scale**" parameter, with larger values giving rise to larger crystals.

An overall scaling factor is provided by "**scale**". Increasing this value will make the marble pattern appear larger.

"mean curvature"

Class **Color**

Synopsis False color shader which shows the `mean curvature' of a surface.

Description The mean curvature of the surface is represented by a false color rainbow. This is done by converting the curvature into an HLS color value. In order to provide different colors for the minimal and maximal values, the range of curvature changes is mapped onto a range hue of only [0, 300], rather than the full [0, 360]. The following table illustrates the color changes:

hue value	*color name*
0	Red
60	Yellow
120	Green
180	Cyan
240	Blue
300	Magenta

The parameter "**min curvature**" specifies the degree of curvature to be mapped onto hue 0, while "**max curvature**" specifies the degree to be mapped onto hue 300. The other colors are assigned using a linear interpolation.

"plain"

Class **Color**

Synopsis A plain, uniform color.

Description A plain, uniform color, specified by argument "**color**".

"solid clouds"

Class **Color**

Synopsis A cloudy appearance.

Description A solid texture giving a cloudy appearance.

The color of the clouds and the background (sky) may be specified in arguments "**clouds color**" and "**background color**", respectively.

The detail or complexity of the texture may be controlled by means of the "**detail**" argument. A value of 1 will give a simple outline of the clouds, while a larger value such as 5 or 6 will give fine detail on the cloud outlines.

An overall scaling factor is provided in argument "**scale**"; increasing this value will make the clouds appear larger.

"solid polka"

Class **Color**

Synopsis A solid polka-dot pattern.

Description A solid texture giving a polka-dot pattern. The texture may be thought of as a three-dimensional lattice of spheres of a given color suspended on a uniform medium with a different color.

The color of the spheres is specified by the argument "**spot color**" and that of the background by the argument "**background color**".

The radius of the spheres is specified by argument "**radius**", and the distance between centers of adjacent spheres by argument "**separation**".

A factor in the range 0.0 to 1.0, which is provided in argument «edge softness», controls the softness of the sphere boundaries. A value of 0.0 gives a hard edge to the spheres, and a value of 1.0 gives a continuous blend of color throughout each sphere.

An overall scaling factor may be specified by argument "**scale**"; increasing this value will make the spheres appear larger.

"surface evaluation"

Class **Color**

Synopsis Color shader that simulates the reflection of a cylinder with longitudinal bands surrounding the surface being considered. Gives good visual clues to the surface's curvature.

Description A traditional method of assessing the curvature of complex surfaces, such as car bodies, has been to place them within a brightly lit cylinder (or half-cylinder) which has longitudinal stripes (or bands). This shader provides the ability to perform this test in software easily.

The color shader is simply attached to the geometry in question (using sensible values for the center, axis and radius) before rendering, and reflections of the imaginary cylinder will be visible on the surface.

The basic color of the cylinder, and the color of the bands, are specified by parameters "**base color**" and "**band color**" respectively.

The size and positioning of the cylinder are controlled by the parameters "**center**", "**axis**" (which should be either X, Y, or Z), and "**radius**".

The total number of bands around the cylinder is given by the parameter "**bands**".

The parameter "**coverage**" determines the ratio of area covered by the bands to the area not covered: a value of zero means only the base color is visible, a value of one means only the band color is visible.

The sharpness of the bands is controlled by "**fuzz**". A value of zero gives very sharply defined edges to the bands; a value of one gives very fuzzy edges.

Finally, parameter "**min angle**" specifies the minimal value of the normal component along the cylinder axis in order for banding to be applied. In other words, it controls how long the virtual cylinder is; smaller values result in a longer cylinder.

Notes:
a) The rays are "reflected" in such a way that the component of reflected ray parallel to the cylinder axis is ignored. It means that all "reflected rays" are perpendicular to the cylinder surface.
b) If part of the surface is placed outside the cylinder, then it is not banded and shows the constant base color. You should ensure your cylinder is big enough and place its center in the middle of the geometry being examined.

"wood rt"

Class **Color**

Synopsis A simple wood pattern with concentric rings of light and dark wood.

Description A solid texture corresponding to a simple wood pattern.

Wood is based on the concept of having a tree trunk centered on a given axis with concentric rings of light and dark wood colors.

Light and dark wood colors are specified in arguments "**light wood color**" and "**dark wood color**" respectively.

The axis of the tree from which the wood is taken is specified by a point that lies on the axis in argument "**point on axis**" and a direction vector pointing along the axis of the tree in argument "**axis direction**".

A factor in the range 0.0 to 1.0 which indicates how "noisy" the rings of the tree appear may be specified in argument "**noise**".

An overall scaling factor is supplied in argument "**scale**"; increasing this value will make the rings of wood appear larger.

"wrapped brick"

Class **Color**

Synopsis A wrapped brick pattern.

Description A wrapped texture giving a simple brick pattern.

All of the bricks have a uniform color which may be specified by argument "**brick color**", and the color of the mortar may be specified by argument "**mortar color**".

The width and height of each brick is specified by the arguments "**brick width**" and "**brick height**" respectively, and the size of the mortar between the bricks (which is uniform between rows and columns of bricks) is specified by argument "**mortar size**".

An overall scaling factor is provided as argument "**scale**"; increasing this value will make the bricks appear larger.

"wrapped checker"

Class **Color**

Synopsis A wrapped checker board pattern.

Description A wrapped texture giving a checker board pattern.

The squares of the pattern take on alternate colors, which may be specified by means of the arguments "**odd color**" and "**even color**".

The size (both width and height) of the squares in the local coordinate system is specified by argument "**scale**".

"wrapped diagonal"

Class **Color**

Synopsis A pattern giving a line across the leading diagonal of the texture space.

Description A color source shader providing a diagonal stripe in texture space.

The size of the pattern is specified by argument "**scale**"; increasing this value will make the pattern appear larger.

The width of the stripe specified as a fraction of the pattern size is given by "**width**".

The color of the stripe and of the background are specified by arguments "**stripe color**" and "**background color**" respectively.

A fuzzy edge is applied to the stripe with a width given by "**fuzz**".

"wrapped filtered image"

Class **Color**

Synopsis A wrapped texture to provide image color mapping with additional color filtering.

Description A wrapped texture that provides image mapping with additional color filtering.

The name of the file containing the image data is provided as a string to parameter "**file name**".

Image color is modulated by the filter color provided by argument "**filter color**".

"wrapped grid"

Class **Color**

Synopsis A grid color pattern.

Description A color shader providing a grid pattern in texture space.

An overall scaling factor for the pattern is given by parameter "**scale**"; increasing this value will make the pattern appear larger.

The width and height of the repeated grid pattern are given by "**width**" and "**height**" respectively, and the width and height of the grid lines are specified by "**grid size**".

The colors of the grid and the background are given by "**grid color**" and "**background color**" respectively.

"wrapped image"

Class **Color**

Synopsis A wrapped texture to provide image color mapping.

Description A wrapped texture that provides image mapping.

The name of the file containing the image data is provided as a string to parameter "**file name**".

"wrapped polka"

Class **Color**

Synopsis A wrapped polka-dot pattern.

Description A wrapped texture giving a polka-dot pattern.

The texture corresponds to a grid of circles of one color against a background of another color.

The color of the spots is specified by the argument "**spot color**" and that of the background by argument "**background color**".

The radius of the circles is specified by argument "**radius**", and the distance between centers of adjacent circles by argument "**separation**".

A factor in the range 0.0 to 1.0 which is provided in argument "**edge softness**" controls the softness of the circle boundaries. A value of 0.0 gives a hard edge to the circles, and a value of 1.0 gives a continuous blend of color throughout each circle.

An overall scaling factor may be specified by argument "**scale**"; increasing this value will make the circles appear larger.

"wrapped roof tiles"

Class **Color**

Synopsis Wrapped color shader simulating various styles of roof tile.

Description This shader which simulates various styles of roof tiles as they are usually seen from ground level (in other words, when looking at the roof surface at an angle of about 45 degrees). Note that the roof tiles pattern that we usually observe is mainly the result of changes in the normals and variable illumination (for example, edges of the tile being in shadow) rather than actual variations in the color of the tile. This shader simulates this using just a surface color pattern.

The "**scale**" parameter allows overall scaling of the pattern.

The "**pattern**" parameter selects the style of tiles which are simulated. The available styles are: flat, scalloped, diamond, hexagonal, classic, split, espana, spanish, roma.

The color of each tile is selected randomly from the range of colors specified by the parameters "**tile color 1**" and "**tile color 2**". If these parameters are the same then the tiles will have a uniform color; if they are different then tile colors will vary across the pattern. "**tile length**" and "**tile width**" control the size of the `top' surface of the tile.

"**tile thickness**" controls how thick the tile is and therefore how much of the edge is visible.

The color of the exposed edge is controlled by the "**edge color**" parameter. Usually this should be darker than the tile colors to simulate the effect of the edge being in shadow.

It is possible to have random color variation within a single tile, for example to simulate dirt on the tile surface. This is controlled by the "**color variation**" parameter. No part of any tile will ever have a color outside the range specified by the tile color parameters.

The value of "color variation" determines how far from the tile's selected base color areas of the tile can deviate. A value of zero will give a uniform color across the entire tile. As the value is increased the variation of color within a tile will increase. With a value of 1.0 a tile with a base color equal to "tile color 1" may contain spots of "tile color 2". "**color variation scale**" controls how the color variation is distributed across the tile. Small values will give a speckled effect whilst larger ones will give larger patches of different colors on the tile.

"wrapped s stripe"

Class **Color**

Synopsis A pattern giving a line along the **s**-axis of the texture space.

Description A color source shader providing a stripe as a line of constant **s** in texture space.

The size of the pattern (including the stripe) is specified by argument "**scale**"; increasing this value will make the pattern appear larger.

The width of the stripe specified as a fraction of the pattern size is given by "**width**".

The colors of the stripe and of the background are specified by arguments "**stripe color**" and "**background color**" respectively.

A fuzzy edge is applied to the stripe with a width given by "**fuzz**".

"wrapped t stripe"

Class **Color**

Synopsis A pattern giving a line along the **t**-axis of the texture space.

Description A color source shader providing a stripe as a line of constant **t** in texture space.

The size of the pattern (including the stripe) is specified by argument "**scale**"; increasing this value will make the pattern appear larger.

The width of the stripe specified as a fraction of the pattern size is given by "**width**".

The colors of the stripe and of the background are specified by arguments "**stripe color**" and "**background color**" respectively.

A fuzzy edge is applied to the stripe with a width given by "**fuzz**".

"wrapped textured brick"

Class **Color**

Synopsis A wrapped textured brick pattern.

Description A wrapped color source texture giving a sophisticated brick pattern.

Each brick has a base color which is randomly assigned from the RGB color range specified by parameters "**brick color 1**" and "**brick color 2**". The bricks are textured by applying to them a coherent variation in color around the base color.

The color of the mortar may be specified by argument "**mortar color**".

The width and height of each brick are specified by the arguments "**brick width**" and "**brick height**" respectively, and the size of the mortar between the bricks (which is uniform between rows and columns of bricks) is specified by argument "**mortar size**".

There is a rough edge where the bricks meet the mortar, the characteristics of which can be controlled through parameters "**rough scale**" and "**rough amplitude**". The former determines the complexity of the roughness along each brick; values less than 1.0 produce a rougher appearance, and values greater than 1.0 a smoother appearance. The latter parameter determines the magnitude of the roughness, with values less than 1.0 corresponding to small variations, and values greater than 1.0 large variations.

An overall scaling factor is provided as argument "**scale**"; increasing this value will make the bricks appear larger.

"chrome 2D"

Class **Reflectance**

Synopsis A chrome-like reflectance using a two-dimensional chrome map.

Description A reflectance model providing a chrome-like effect which generates a reflection pattern from a two dimensional array of colors.

The map is projected onto surfaces using a spherical mapping. Reflectance coefficients are provided and they allow specification of the amount of ambient, diffuse, specular and chrome light reflected, corresponding to arguments "**ambient factor**", "**diffuse factor**", "**specular factor**" and "**chrome factor**" respectively.

The sharpness of the specular reflection highlights can be controlled by means of argument "**roughness**". The reflection is made sharper with small values for the roughness, such as 0.1. Larger values, such as 10.0, decrease the specular fall-off.

This model is suitable for shiny or highly polished chrome-like materials.

"conductor"

Class **Reflectance**

Synopsis Physically accurate metallic simulation using ray tracing, supporting reflection.

Description A reflectance model that supports secondary mirrored views through ray tracing. Fresnel filtering is incorporated for accurate modeling of light interaction on the surfaces of objects.

Reflectance coefficients are provided to allow specification of the amount of ambient, diffuse and specular light components, and the contribution made by light reflected in the mirror direction, corresponding to arguments "**ambient factor**", "**diffuse factor**", "**specular factor**" and "**mirror factor**" respectively.

The sharpness of specularly reflected highlights can be controlled by means of argument "**roughness**". The reflection is made sharper with small values for the roughness, such as 0.1. Larger values, such as 10.0, decrease the specular fall-off.

The conductor's indices of refraction for red, green and blue light are given by the arguments "**refraction red**", "**refraction green**" and "**refraction blue**" respectively.

Similarly, "**absorption red**", "**absorption green**" and "**absorption blue**" give the absorption coefficients for red, green and blue light respectively.

This model is appropriate for simulating metallic surface finishes accurately.

The default values for the refraction and absorption coefficients correspond to those for silver.

The appropriate values for other conductors can be obtained from published data in handbooks of optical constants: a sample of the properties of some common metallic looking materials is given in the next table.

Conductor	**Color**			**Refraction**			**Absorption**		
Materials	*red*	*green*	*blue*	*red*	*green*	*blue*	*red*	*green*	*blue*
aluminium	0.900	0.922	0.924	1.801	0.863	0.592	8.002	6.383	5.341
chromium	0.640	0.669	0.695	3.749	2.869	1.837	4.364	4.440	4.010
cobalt	0.693	0.649	0.612	2.286	1.973	1.694	4.358	3.697	3.194
copper	0.929	0.613	0.540	0.302	1.051	1.168	3.916	2.582	2.338
gold	0.952	0.697	0.389	0.186	0.590	1.518	3.756	2.297	1.897
graphite	0.365	0.299	0.291	2.925	2.663	2.630	1.737	1.332	1.286
mercury	0.780	0.779	0.773	2.250	1.426	1.034	5.515	4.464	3.748
nickel	0.663	0.596	0.520	2.131	1.726	1.628	3.932	3.105	2.581
palladium	0.743	0.690	0.649	1.837	1.582	1.385	4.529	3.704	3.180
platinum	0.683	0.629	0.581	2.489	2.054	1.814	4.389	3.581	3.067
silver	0.973	0.955	0.908	0.138	0.128	0.157	4.377	3.174	2.344
tungsten	0.508	0.492	0.467	3.695	3.453	3.338	2.830	2.714	2.498

"constant"

Class **Reflectance**

Synopsis Constant reflectance.

Description A reflectance model providing a constant color. This model ignores the effect of all light sources and gives the same result as an environment containing a single, white, ambient source with intensity 1.

"dielectric"

Class **Reflectance**

Synopsis Physically accurate glass-like simulation using ray tracing, supporting reflection.

Description A reflectance model that supports an accurate simulation of dielectric (glass-like) materials that have both reflective and transmissive properties. Secondary mirrored and transmitted views are incorporated by the use of ray tracing. Fresnel filtering is incorporated for the specular reflection, the mirrored view and the transmitted view, for accurate modeling of light interaction on the surfaces of objects.

Reflectance coefficients are provided to allow specification of the amount of ambient, diffuse and specular light components, and the contribution made by light from the mirror and transmission directions, corresponding to arguments "**ambient factor**", "**diffuse factor**", "**specular factor**", "**mirror factor**" and "**transmission factor**" respectively.

The sharpness of the specular reflection highlights can be controlled by means of argument "**roughness**". The reflection is made sharper with small values for the roughness, such as 0.1. Larger values, such as 10.0, decrease the specular fall-off.

The dielectric's index of refraction for all wavelengths of light is given by the argument "**refraction**". The default value for the refractive index corresponds to that for glass. The appropriate value for other dielectrics can be obtained from published data in handbooks of optical constants.

This model is appropriate for simulating glass surface finishes.

"environment"

Class **Reflectance**

Synopsis Environment mapped reflectance.

Description A reflectance model providing environment mapping.

Reflectance coefficients are provided as arguments which allow specification of the ambient ("**ambient factor**"), diffuse ("**diffuse factor**"), specular ("**specular factor**") and environment ("**environment factor**") terms.

The sharpness of the specular reflection highlights can be controlled by means of argument "**roughness**". The reflection is made sharper with small values for the roughness, such as 0.1. Larger values, such as 10.0, decrease the specular fall-off.

A scaling factor for the area of sampling in the global environment map is given by "**angle scale**"; a value of 0 will cause point sampling, a value greater than the default of 1 will cause 'diffuse' environment reflections.

This model is suitable for reflective materials such as shiny metals.

"glass"

Class **Reflectance**

Synopsis Approximation of glass-like materials using ray tracing, supporting reflection and refraction.

Description A reflectance model that supports an approximation of glass-like materials that have both reflective and transmissive properties.

Secondary mirrored and transmitted views are incorporated by the use of ray tracing.

Reflectance coefficients are provided as arguments which allow specification of the amount of the specular light component ("**specular factor**", and the contribution made by light from the mirror ("**mirror factor**") and transmission ("**transmission factor**") directions.

The sharpness of the specular reflection highlights can be controlled by means of argument "**roughness**". The reflection is made sharper with small values for the roughness, such as 0.1. Larger values, such as 10.0, decrease the specular fall-off.

The index of refraction for all wavelengths of light is given by the argument "**refraction**". The default value for the refractive index corresponds to that for glass.

The appropriate value for other materials can be obtained from published data in handbooks of optical constants.

This model is appropriate for approximating glass surface finishes.

"matte"

Class **Reflectance**

Synopsis Dull matte-like reflectance.

Description A reflectance model providing a dull matte appearance.

Reflectance coefficients are provided to allow specification of the amount of ambient and diffuse light reflected in arguments "**ambient factor**" and "**diffuse factor**" respectively.

This model is suitable for non-glossy materials such as brick or fabric.

"metal"

Class **Reflectance**

Synopsis Shiny metallic reflectance.

Description A reflectance model providing a specular metallic appearance.

Reflectance coefficients are provided to allow specification of the amount of ambient and specular light reflected in arguments "**ambient factor**" and "**specular factor**" respectively.

The sharpness of specular reflections can be controlled by means of the "**roughness**" argument. The reflection is made sharper with small roughness values (less than 1), such as 0.1. Larger values, such as 10.0, decrease the specular fall-off.

This model is suitable for most metallic materials such as steel or brass.

"mirror"

Class **Reflectance**

Synopsis Approximation of mirror reflecting materials using ray tracing, supporting reflection.

Description A reflectance model that supports secondary mirrored views through ray tracing.

Reflectance coefficients are provided as arguments which allow specification of the amount of ambient light ("**ambient factor**"), diffuse light ("**diffuse factor**") and specular light ("**specular factor**") components, and the contribution made by light reflected in the mirror direction ("**mirror factor**").

The sharpness of the specular reflection highlights can be controlled by means of argument "**roughness**". The reflection is made sharper with small values for the roughness, such as 0.1 or less. Larger values, such as 10.0, decrease the specular fall-off.

This model is appropriate for representing mirror-like surface finishes.

"multilayer paint"

Class **Reflectance**

Synopsis Empirical simulation of multi-layer paint surfaces.

Description A reflectance that simulates the effects of multi-layer paint used in the automobile industry.

The reflectance effect is achieved by a combination of several difference reflectance model into a single shader. The shader is build using a combination of other reflectance shaders. This means that there will be parameter duplications. This is necessary to allow for individual control of each layer's properties such as the reflection from the lacquer layer and the metallic layer.

The exact meaning of each of the arguments is given below:

"**lacquer specular factor**" Specular factor for the lacquer layer.

"**lacquer transmission factor**" Transmission factor for the lacquer layer. The underlying color is filtered by this value.

"**lacquer mirror factor**" Mirror factor for the lacquer layer.

"**lacquer roughness**" Roughness of the lacquer layer.

"**lacquer refraction**" Refraction index of the lacquer layer.

"**metallic layer factor**" Contribution of the metallic layer. The inverse determines the contribution from the base layer.

"**metallic flakes**" Select the 'metal' of the flakes.

"**metallic roughness**" Sharpness of the flakes' reflection. This is similar to the 'conductor' shader 'roughness' parameter.

"**metallic scale**", "**metallic amplitude**", "**metallic detail**", "**metallic sharpness**" are taken from 'rough' to simulate micro facet surfaces.

"**base ambient factor**" Ambient factor for the base layer.

"**base diffuse factor**" Diffuse factor for the base layer.

"phong"

Class **Reflectance**

Synopsis Phong-style reflectance.

Description A reflectance model conforming with the popular Phong model in which reflections are greatest in the ``mirror'' direction of a surface opposite the viewing direction with respect to the surface normal.

Reflectance coefficients are provided to allow specification of the amount of ambient, diffuse and specular light reflected, corresponding to arguments "**ambient factor**", "**diffuse factor**" and "**specular factor**" respectively.

The sharpness of the specular reflection highlights can be controlled by means of argument "**exponent**". The reflection is made sharper with large values of the exponent, such as 10.0. Small values, such as 1.0, decrease the specular fall-off.

The color of the specular highlights is filtered by the color argument "**specular color**".

This model is suitable for shiny or highly polished materials such as ceramic or glass.

"plastic"

Class **Reflectance**

Synopsis Glossy plastic-like reflectance.

Description A reflectance model providing a specular effect which is similar to the Phong model.

Reflectance coefficients are provided to allow specification of the amount of ambient, diffuse and specular light reflected, corresponding to arguments "**ambient factor**", "**diffuse factor**" and "**specular factor**" respectively.

The sharpness of the specular reflection highlights can be controlled by means of argument "**roughness**". The reflection is made sharper with small values for the roughness, such as 0.1. Larger values, such as 10.0, decrease the specular fall-off.

The color of the specular highlights is filtered by the color argument "**specular color**".

This model is suitable for shiny or highly polished materials such as plastic or varnished surfaces.

"shadow catcher"

Class **Reflectance**

Synopsis Allows shadows to fall upon hidden geometry in a scene.

Description Usually the supplied geometry will follow some detail depicted by a background image. This then allows objects that have been placed in the scene to cast shadows that appear to interact with the background in a natural way.

The "**catch**" parameter allows the shadow analysis to be controlled: If you set *None*, shadow analysis is not performed but the shadow catcher primitive is shaded directly with the background followed by execution of any specified reflectance shader. If you set *Catch all*, shadows are cast by the scene geometry and the absence of light is considered as total shadow. For example, a region outside a spot light cone is determined to be in total shadow. If you set *Shadows only*, shadow analysis is performed as described above but in this mode only shadow created by geometry is considered to be a true shadow.

"**shadow color**" specifies a filter color to be applied to the shadow. Use this parameter to tint a shadow a desired color, for example to match the ambient lighting conditions. Note that if any ambient lights are illuminating the scene then they only affect the shadowed area and not the shadow catcher geometry as a whole. The effects of ambient lighting are added to the shadow region before any filtering by shadow color is performed.

Any ambient lighting contributions to the shadow region are also clamped to ensure that this region is never brighter than its surroundings. "**shadow brightness**" limits the darkest value that a shadow can be, where 0.0 is total shadow (i.e., black) and 1.0 is total illumination. This threshold value is applied as a limiting value to the total shadow at a point. The total shadow is the sum of the individual shadow contributions from each light source.

"**reflectance**" a reflectance class shader that may be used as input to the shader to provide the geometry with the reflectance properties that match the scenery it represents. For example, a reflectance shader may be used so that background covered by a shadow catcher responds to the lighting conditions in a natural way.

"translucency"

Class **Reflectance**

Synopsis Simulates a non-reflective translucent surface lit from behind.

Description A reflectance model providing a translucent, backlit appearance.

The diffuse lighting component is obtained from lights on opposite side of the surface to the viewer, i.e., surface normal is effectively reversed. Reflectance co-efficients allowing specification of this amount of ambient light and degree of translucency are supplied by arguments "**ambient factor**" and "**translucency factor**" respectively. This model is suitable for non-glossy, translucent materials such as dull, translucent plastic.

"translucent plastic"

Class **Reflectance**

Synopsis Glossy plastic-like reflectance with translucency.

Description A reflectance model providing a specular effect which is similar to the Phong model with translucency.

Reflectance is as for the "plastic" shader but an additional diffuse term is added which represents the diffuse contribution from lights on the reverse side of the surface. The degree of translucency is controlled by the argument "**translucency factor**".

Reflectance co-efficients are provided which allow specification of the amount of ambient, diffuse and specular light reflected, corresponding to arguments "**ambient factor**", "**diffuse factor**" and "**specular factor**" respectively.

The sharpness of the specular reflection highlights can be controlled by means of argument "**roughness**". The reflection is made sharper with small values for the roughness, such as 0.1. Larger values, such as 10.0, decrease the specular fall-off. The color of the specular highlights is filtered by the color argument "**specular color**". This model is suitable for shiny or highly polished translucent materials such as translucent plastic.

"wrapped anisotropic"

Class **Reflectance**

Synopsis Approximation of surface with parallel scratches or ridges, such as brushed metal.

Description Anisotropic reflectance shader simulates the directional reflectance properties of surfaces covered in parallel scratches or grooves. Useful for simulating polished metal or grooved surfaces.

Surface scratches are approximated by thin cylinders. The degree of anisotropy of the surface can be controlled by varying the distance between the cylinders using the "**cylinder distance**" parameter. A distance of 0.0 gives isotropic (normal) reflection, and distance of 2.0 gives maximum anisotropy. The height of the floor across the cylinders may also be specified with the "**floor height**" parameter. A value of 0.0 gives no floor, a value of 1.0 gives a completely flat, isotropic surface.

The surface direction is determined from the axes of the texture space assigned to the primitive. Cylinders are aligned at 90 degrees to the **s**-axis of the texture co-ordinate system (i.e., aligned with the **t**-axis for a right angled texture co-ordinate system). It is important to assign the correct texture co-ordinates to achieve the desired effect. The effect of default texture co-ordinates is undefined.

Reflectance co-efficients are provided as arguments which allow specification of the amount of ambient light ("**ambient factor**"), diffuse light ("**diffuse factor**") and specular light ("**specular factor**") components as for the plastic shader. The sharpness of the specular highlights can be controlled using the "**roughness**" parameter and the color may be controlled using the "**specular color**" parameter.

"wrapped circular anisotropic"

Class **Reflectance**

Synopsis Approximation of surface with many small areas of concentric rings of scratches or ridges repeated over the surface.

Description Anisotropic reflectance shader that simulates the directional reflectance properties of surfaces covered in a repeating pattern of concentric scratches or grooves.

Surface scratches are approximated by thin cylinders. The degree of anisotropy of the surface can be controlled by varying the distance between the cylinders using the "**cylinder distance**" parameter. A distance of 0.0 gives isotropic (normal) reflection, and distance of 2.0 gives maximum anisotropy. A distance of greater than 2.0 gives a flat floor between cylinders which reflects light isotropically. The height of the floor across the cylinders may also be specified with the "**floor height**" parameter. A value of 0.0 gives no floor, a value of 1.0 gives a completely flat, isotropic surface.

The center of the circular pattern is defined to be at (0.5, 0.5) in texture space.

The horizontal and vertical frequency of the pattern may be controlled using the "**width**" and "**height**" parameters. Setting either of these to zero (default) prevents the pattern from repeating. The pattern is also aligned with the **s** and **t** texture coordinate axes and hence texture coordinates or a texture space must be supplied to ensure the correct results. Texture coordinates may also be used to control the frequency, orientation and alignment of the circular patches. If the texture coordinates are supplied by a texture space shader then primitive need not be triangulated, but the behavior of the shader along surfaces with constant **s** or **t** is undefined.

Reflectance co-efficients are provided as arguments which allow specification of the amount of ambient light ("**ambient factor**"), diffuse light ("**diffuse factor**") and specular light ("**specular factor**") components as for the plastic shader. The sharpness of the specular highlights can be controlled using the "**roughness**" parameter and the color may be controlled using the "**specular color**" parameter.

"wrapped mirror map"

Class **Reflectance**

Synopsis Approximation of mirror reflecting materials using ray tracing, supporting reflection with degree of reflectance determined by red component of an image map.

Description A reflectance model that supports secondary mirrored views through ray tracing, with degree of reflectance determined by the red component of a texture image.

The image is specified via the name of a file ("**file name**"). Reflectance coefficients are provided as arguments which allow specification of the amount of ambient light ("**ambient factor**"), diffuse light ("**diffuse factor**") and specular light ("**specular factor**") components, and the contribution made by light reflected in the mirror direction ("**mirror factor**").

The sharpness of the specular reflection highlights can be controlled by means of argument "**roughness**". The reflection is made sharper with small values for the roughness, such as 0.1 or less. Larger values, such as 10.0, decrease the specular fall-off.

This model is appropriate for representing mirror-like surface finishes in which the degree of reflectance is variable across the surface.

"eroded"

Class **Transparency**

Synopsis Surface erosion.

Description A transparency shader operating as a stencil that creates the illusion of erosion on a surface.

The scale of the erosions is given by parameter "**scale**"; increasing this value will increase the size of the erosions.

The degree of erosion is controlled by the "**coverage**" parameter, this being a value in the range zero to one where one denotes full coverage (no erosions) and zero no coverage (complete erosion).

The edges of the erosions are given a fuzziness whose value is based upon the "**fuzz**" parameter, which should be in the range zero to one, with larger values producing softer edges to the erosions.

"glow"

Class **Transparency**

Synopsis Transparency shader that can be used to create effect of a region of glow around a light source.

Description Transparency shader which varies coverage (alpha) value depending on the angle between surface normal and view direction. Useful for creating glow effects when applied to spheroid surfaces (such as spheres or ellipsoids). Such spheres are placed around a light source, this transparency shader is applied, and the effect is of a spherical glow around the light source.

The coverage can be set for the center ("**center coverage**") and the edge ("**edge coverage**"). If a sphere is used with a high central coverage and near-zero edge coverage the effect is of a gradually diminishing glow around the light source. If edge coverage is higher than central coverage the effect is similar to a bubble around the light source.

"label"

Class **Transparency**

Synopsis A transparency shader specific for labeling.

Description A transparency shader which gives you specific controls over an image labeled over a surface.

The parameter "**inside coverage**" determines the ratio of area covered by the label to the area not covered: a value of zero means the label is completely transparent. "**outside coverage**" determines the ratio of area covered by the background to the area not covered: a value of zero means the background is completely transparent.

The sharpness of the label edges is controlled by "**u fuzz**" and "**u fuzz**". A value of zero gives very sharply defined edges, a value of one gives very fuzzy edges.

"plain"

Class **Transparency**

Synopsis A plain, uniform transparency.

Description The transparency is specified as a color *filter* value by argument "**color**". This enables colored transparency and translucency, such as colored glass, to be simulated. Each of the red, green and blue components of the "color" argument should lie in the range 0.0 to 1.0, where a value of 0.0 corresponds to completely transparent, and 1.0 to completely opaque.

"plain coverage"

Class **Transparency**

Synopsis A plain, uniform transparency.

Description Applies a constant alpha transparency value to a material. Set the parameter "**coverage**" with a value and the material will be rendered with an alpha transparency equal to that value, e.g., set coverage to 0.5 and the object will appear 50 transparent. Set to 1.0 for opaque and 0.0 for invisible. The transparency is constant across the object.

"wrapped checker"

Class **Transparency**

Synopsis A wrapped checker transparency pattern.

Description A wrapped texture giving a checker transparency pattern.

The degree of transparency of the squares is specified by means of the arguments "**odd coverage**" and "**even coverage**".

The size (both width and height) of the squares in the local coordinate system is specified by argument "**size**".

"wrapped grid"

Class **Transparency**

Synopsis A grid transparency pattern.

Description A transparency shader providing a grid pattern in texture space. The grid will appear opaque, with holes visible between the grid lines.

An overall scaling factor for the pattern is given by parameter "**scale**"; increasing this value will make the pattern appear larger.

The width and height of the repeated grid pattern are given by "**width**" and "**height**" respectively, and the grid lines width and height are specified by "**grid size**".

A transparency value as a scalar quantity can be specified for the grid pattern with parameter "**transparency**", where a value of zero indicates opaque, and a value of one fully transparent.

"wrapped image"

Class **Transparency**

Synopsis A wrapped texture to provide image transparency mapping.

Description A wrapped texture that provides image mapping for transparency shading.

The name of the file containing the image data is provided as a string to parameter "**file name**". The file must contain image data in a format for which an appropriate image driver is currently installed.

"wrapped mask"

Class **Transparency**

Synopsis Wrapped image in which the red channel is interpreted as the transparency alpha value.

Description The image/texture file specified is used as a mask by interpreting the 'red' channel of the file as the transparency alpha value: i.e., a value of 0.0 is transparent, and a value of 1.0 opaque.

Values between 0.0 and 1.0 allow the background to be seen 'through' the foreground (i.e., a translucent effect is generated). The "wrapped mask" transparency shader would often be used as transparency shader associated with a "decal" shader.

The name of the file containing the image data is provided as a string to parameter "**file name**". The file must contain image data in a format for which an appropriate image driver is currently installed.

"wrapped square"

Class **Transparency**

Synopsis A square region of transparency.

Description A transparency shader that provides a single square aligned with the texture space.

There is a soft edge to the square, expressed as a 'fuzz' factor which encodes a value in the range 0 to 1. Fuzz can be specified separately for the s and t dimensions. The alpha value inside and outside the square are given by "**inside coverage**" and "**outside coverage**" respectively.

The degree of edges blurring can be controlled by the "**fuzz**" parameter. The parameter range is 0.0 to 1.0. 0.0 means no fuzz, i.e. edges are razor sharp. 1.0 gives maximal blurring, i.e. the blurred area width is equal to the half of the square size.

"casting"

Class **Displacement**

Synopsis A cast metal-like finish.

Description A displacement providing an irregular casting pattern. This pattern is derived from two superimposed noise functions, providing displacements and indentations, the latter of which being used to scale the displacements.

The magnitude of the surface displacements, expressed as a factor, typically in the range 0.0 to 1.0, is specified by argument "**casting amplitude**", and the amplitude of the indentations that superimpose the displacements is specified by "**dented amplitude**".

An integer value which controls the detail or complexity of the texture is specified by argument "**detail**", such that a value of 1 will give a simple perturbation, while a larger value such as 5 or 6 will produce fine detail in the pattern.

An overall scaling factor may be supplied in argument "**scale**"; increasing this value will make the surface perturbations appear larger.

Scaling of the indentations can be controlled separately by the argument "**dented scale**".

The relative contributions made by the displacements and indentations are controlled by argument "**dented threshold**". This value should vary between 0.0 and 1.0, with a value of 0.0 causing maximum interaction.

This displacement is suitable for any surface where an irregular roughness is required.

"leather"

Class **Displacement**

Synopsis Models surface texture of leather using a solid displacement texture.

Description Displacement shader which mimics the surface detail of leather using a solid texture. The basic pattern is a series of "cells" of unit size which vary from square to irregular convex shape using the parameter "irregularity" which varies from 0.0 to 1.0.

The height of each cell is controlled using "**cell amplitude**" and the smoothness of the edges is controlled by a smooth step function with the minimum and maximum values set using "**smooth min**" and "**smooth max**" parameters. The shape of the grooves between cells can also be controlled.

The parameter "**curve amplitude**" determines how curved the edges of the cells are, the parameter "**curve frequency**" determines how wriggly the edges are, with a value of 1.0 indicating one cycle per cell. The detail in the grooves between cells can be controlled using "**curve detail**". This varies from 1 to 10 and gives the grooves an increasingly creased appearance.

The bumpiness of the surface can be controlled using two sets of "rough" and "fold" parameters. These add bumps to the surface, the amplitude controlled by "**rough amplitude**" and "**fold amplitude**", the detail controlled by "**rough detail**" and "**fold detail**", and the frequency by "**rough frequency**" and "**fold frequency**". The "rough" parameters refer to high frequency bumps which occur within cells, and the "fold" parameters refer to widespread undulating pattern used to simulate wrinkles in the material.

The number of cells per unit square is determined using the "**scale**" parameter. When set to one there is on average one cell per unit square. The amplitude of cells scales inversely with the scale factor, hence the more cells per unit volume the lower the amplitude of the cell.

"rough"

Class **Displacement**

Synopsis A rough, undulating finish.

Description A displacement providing a rough, cast metal-like finish.

The appearance of the roughness can be controlled precisely. The magnitude of the surface perturbations, expressed as a factor, typically in the range 0.0 to 1.0, is specified by argument "**amplitude**".

An integer value which controls the detail or complexity of the texture is specified by argument "**detail**", such that a value of 1 will give a simple perturbation, while a larger value such as 5 or 6 will give fine detail of the roughness.

Argument "**sharpness**" is an integer value which controls the sharpness of the perturbations. A value of 1 will give abrupt, sharp changes between the peaks and troughs of the displacements, while a larger value such as 3 or 4 will give smoother transitions.

An overall scaling factor may be supplied in argument "**scale**"; increasing this value will make the surface perturbations appear larger. This displacement is suitable for any surface where an uneven appearance is required.

"wrapped bump map"

Class **Displacement**

Synopsis A bump map displacement derived from an image file.

Description A wrapped displacement shader providing a bump map from an image file in a supported format.

The file name for the image file is supplied as the "**file name**" parameter.

The maximum amplitude for displacements is specified by parameter "**amplitude**". Larger values will accentuate the differences between peaks and troughs of the bump map. Values less than 0.0 will reverse its direction (peaks will become troughs, and vice versa).

"wrapped dimple"

Class **Displacement**

Synopsis A regular dimpled appearance.

Description A wrapped displacement providing a regular dimpled appearance to a material. This displacement may be thought of as a grid of spheres which protrude above a material's surface.

The distance between the centers of adjacent spheres is specified by argument "**separation**", and the radius of a sphere is set with argument "**radius**".

The distance beneath the surface of the centers of the spheres is specified by argument "**center depth**". This distance must be non-negative, and should be no greater than the radius of the spheres.

A value between 0.0 and 1.0, which controls the size of a blend between the sphere and the surface, is specified by argument "**blend**", for which a value of 0.0 indicates no blend, and a value of 1.0 maximum blend.

An overall scaling factor can be specified by argument "**scale**"; increasing this value will make the dimples appear larger.

"wrapped knurl"

Class **Displacement**

Synopsis A knurled appearance.

Description A displacement shader providing a knurled pattern similar to grips used on tool handles.

With default scaling, each unit square in texture space contains a pyramidal displacement. A blend is applied to the edges of the pyramid with the size defined by parameter "**blend**". This value should lie in the range 0.0 to 1.0, where 0.0 denotes no blend (the displacements correspond to pyramids) and 1.0 denotes maximum blend (the displacements correspond to cones).

Two scaling factors may be applied. The "**amplitude**" parameter applies a scale to the height of the displacements. A value between 0.0 and 1.0 will flatten the indentations, and values greater than 1 will accentuate the relative differences in height over the indentation. Values less than 0.0 will reverse the direction of the indentations.

An overall scaling factor can be specified by argument "**scale**"; increasing this value will make the indentation pattern appear larger.

"wrapped leather"

Class **Displacement**

Synopsis Models surface texture of leather using a solid displacement texture.

Description Displacement shader which mimics the surface detail of leather using a solid texture. The basic pattern is a series of "cells" of unit size which vary from square to irregular convex shape using the parameter "irregularity" which varies from 0.0 to 1.0.

The height of each cell is controlled using "**cell amplitude**" and the smoothness of the edges is controlled by a smooth step function with the minimum and maximum values set using "**smooth min**" and "**smooth max**" parameters. The shape of the grooves between cells can also be controlled.

The parameter "**curve amplitude**" determines how curved the edges of the cells are, the parameter "**curve frequency**" determines how wriggly the edges are, with a value of 1.0 indicating one cycle per cell. The detail in the grooves between cells can be controlled using "**curve detail**". This varies from 1 to 10 and gives the grooves an increasingly creased appearance.

The bumpiness of the surface can be controlled using two sets of "rough" and "fold" parameters. These add bumps to the surface, the amplitude controlled by "rough amplitude" and "**fold amplitude**", the detail controlled by "**rough detail**" and "**fold detail**", and the frequency by "**rough frequency**" and "fold frequency". The "rough" parameters refer to high frequency bumps which occur within cells, and the "fold" parameters refer to widespread undulating pattern used to simulate wrinkles in the material.

The number of cells per unit square is determined using the "**scale**" parameter. When set to one there is on average one cell per unit square. The amplitude of cells scales inversely with the scale factor, hence the more cells per unit volume the lower the amplitude of the cell.

"wrapped rough"

Class **Displacement**

Synopsis A wrapped rough, cast metal-like finish.

Description A wrapped displacement providing a rough, cast metal-like finish. The appearance of the roughness can be controlled precisely. The magnitude of the surface perturbations, expressed as a factor, typically in the range 0.0 to 1.0, is specified by argument "**amplitude**".

An integer value which controls the detail or complexity of the texture is specified by argument "**detail**", such that a value of 1 will give a simple perturbation, while a larger value such as 5 or 6 will give fine detail of the roughness.

Argument "**sharpness**" is an integer value that controls the sharpness of the perturbations. A value of 1 will give abrupt, sharp changes between the peaks and troughs of the displacements, while a larger value such as 3 or 4 will give smoother transitions.

An overall scaling factor may be supplied in argument "**scale**"; increasing this value will make the surface perturbations appear larger.

This displacement is suitable for any surface where an uneven appearance is required.

"wrapped tread plate"

Class **Displacement**

Synopsis A tread plate pattern.

Description A wrapped displacement shader providing a regular tread plate pattern to a material. The indentations may be thought of as cylinders with rounded (spherical) ends that protrude above the surface of the material. The fractional radius of the cylinders and spheres is given by "**radius**", which is a fractional value in the range 0.0 to 1.0. Small radii produce longer and thinner indentations.

A blend may be applied to the edges of the indentations where they meet the flat surrounding surface. The size of this blend is specified by "**blend**", which corresponds to a fractional value in the range 0.0 to 1.0. A value of 0.0 produces no blend, and a value of 1.0 results in the whole of the indentation being a blend region.

Two scaling factors may be applied. The "**amplitude**" parameter applies a scale to the height of the displacements. A value between 0.0 and 1.0 will flatten the indentations, and values greater than 1.0 will accentuate the relative differences in height over the indentation. Values less than 0.0 will reverse the direction of the indentations. An overall scaling factor can be specified by argument "**scale**"; increasing this value will make the indentation pattern appear larger.

5.3 Textures

During the shading process, the material shader components calculate shading values by accessing a set of global input variables that completely characterize the surface point to be shaded.

This information includes the location of the point being shaded in both the object's local coordinate system, and the world (global) coordinate system. Also provided are a geometric surface normal vector, a shading normal vector, and an orientation vector with respect to the view direction. Any or all of these geometric values may be employed by a material shader component in order to perform its calculations. Material shader components that calculate their results based upon the shading point's position in space are able to generate arbitrarily complex and coherent patterns. Examples of these are a wood or a marble color source shader.

Shader elements with this property can be classified according to the nature of the positional information upon which the calculations are based.

The first type of material shader element is generally known as a ***solid texture***. Solid textures use the *three-dimensional coordinate system* of the shading point from which to derive their results. Consequently, they create the appearance of a primitive having been carved out of a solid block of the particular material, and the visual characteristics at any point on its surface are those taken on by the solid material at the places that coincide with the object's exterior.

The second type is known as a ***wrapped texture***. Unlike solid textures, which operate within a three-dimensional coordinate system, wrapped textures operate within a *two-dimensional coordinate system* known as ***texture space***.

A simple example should help you understand the difference between solid and wrapped textures. In the following pictures, the same vase model has been shaded using similar color source components of their materials, but using a solid texture in the first case (a), and a wrapped texture in the second (b).

a) solid texture **b) wrapped texture**

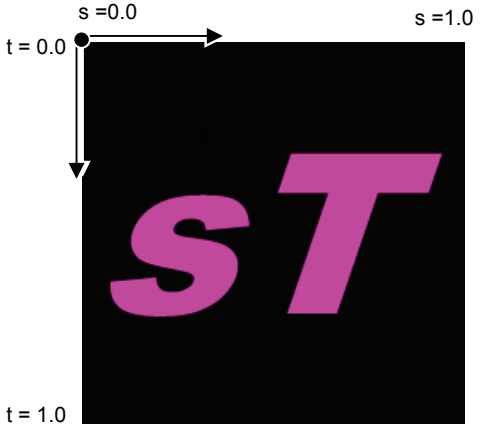
The vase in picture a) seems to be carved out of a solid piece of material with these properties. Where the surface of the vase passes through the center of one of the spheres, the dots appear larger, whereas in the places where the surface of the vase passes through the edge of one of the spheres, the dots appear smaller. Furthermore, in some places distortions in the pattern are apparent, while in others large areas of the surface may be void of any circular patterns where these areas do not coincide with any of the spheres. Compare this with picture (b) that uses a wrapped texture. In this case, the pattern corresponds to a grid of circles against a blue background. This pattern may be thought of as having been printed on a sheet of paper which has first been wrapped around the vase and then shrunk so that it fits tightly and without distortion. In this image, the circles appear uniformly and distributed around the surface.

The Texture Coordinate System

Texture space is a two-dimensional coordinate system denoted by *(s; t)*.

In general, when a texture space is applied to an object and viewed from the outside, the axes formed by the increasing s-direction, t-direction and the surface normal vector will form a left-handed system. Texture space is configured in a such manner that, when a pattern (such as image data) is applied, the origin of the pattern is located at its *top-left corner* with s increasing to the right of the image, and t to the bottom. Also, many texture space shaders operate within the unit square defined by 0 _ s; t _ 1, and the texture pattern will repeat outside of this range.

A simple texture pattern and its associated texture coordinate system are illustrated in the following picture.



Texture Space Shaders Description

The specification of the texture space may be changed for different objects, and it is defined by means of ***texture space shaders***.

Texture space shaders have the same geometric positional information available to them just as material components, and they use this information to derive the corresponding texture space coordinates. These texture space coordinates may then be used by any wrapped textures that form part of the material definition. Texture spaces are assigned to objects in the same way as materials.

***Important Note**: **The texture space function itself will only be called if any of the objects to which it is attached have materials that include one or more wrapped textures.** In the case where an object uses a material with a component which operates in texture space, and no texture space is attached to the primitive, then a default texture coordinate system will be employed.*

In the next pages we will describe the supplied texture space shaders.

"arbitrary plane"

Class **Texture Space**

Synopsis A planar mapping of arbitrary orientation.

Description A texture space in which all points are mapped onto an arbitrary plane.

The plane is specified in terms of a point which lies in the plane as argument "**origin**", a normal vector perpendicular to the plane as argument "**normal vector**", and an orientation expressed as an ``up'' vector passed in argument "**up vector**". The point passed as "origin" is used for the origin of textures such that the texture origins appear at this point.

An overall scaling factor can be passed as argument "**scale**". Increasing this value will have the effect of increasing the scale of all wrapped textures which use this texture space.

The aspect ratio of the texture space, defined as one unit of its height divided by one unit of its width, may be set by the value passed to argument "**aspect ratio**". This is interpreted such that the horizontal scale is equal to the value passed as argument "scale", and the vertical scale is equal to the product of this value and the value passed for argument "aspect ratio".

"auto axis"

Class **Texture Space**

Synopsis Automatic selection from one of the three co-ordinate axes.

Description A texture space that selects one of the three coordinate axes (x- y- and z-axes) whose plane is most closely aligned with the surface at each point. The points in space are then mapped onto this plane.

An overall scaling factor is provided by argument "**scale**".

Increasing this value will have the effect of increasing the scale of all *wrapped* textures which use this mapping.

"cylindrical"

Class **Texture Space**

Synopsis A cylindrical projection.

Description A texture space in which all points in space are mapped onto a cylinder.

The cylinder is specified in terms of a point on its axis in argument "**center point**", and a direction vector along the axis in argument "**axis direction**".

The origin of wrapped textures is specified by means of argument "**origin**".

Scaling factors are provided in the direction *around* the axis in argument "**scale around axis**", and in the direction *along* the axis in argument "**scale along axis**".

"spherical"

Class **Texture Space**

Synopsis A spherical projection.

Description A texture space in which all points in space are mapped onto a sphere.

The sphere is specified in terms of its center point, passed as argument "**center point**", and a point which corresponds to the origin of wrapped textures, passed as argument "**origin**".

The orientation is specified by an "up" vector employed as argument "**axis direction**", which corresponds to the upward pointing axis of the sphere.

Scaling factors are provided, one in the latitudinal direction ("**latitude scale**") and one in the longitudinal direction ("**longitude scale**").

"uv"

Class **Texture Space**

Synopsis Texture space derived from the parametric co-ordinate system.

Description A texture space shader that assigns the parametric coordinate system to the texture space.

Independent scaling factors are provided for the u- and v-coordinate directions in arguments "**u scale**" and "**v scale**" respectively.

"uv label"

Class **Texture Space**

Synopsis Texture space derived from the parametric co-ordinate system. Very useful for labeling.

Description A texture space shader that assigns the parametric coordinate system to the texture space.

Independent scaling factors are provided for the u- and v-coordinate directions in arguments "**u scale**" and "**v scale**" respectively.

It is also possible to change the "**angle**" and specify the origin in the uv space by modifying "**x origin**" and "**y origin**" arguments.

"x plane"

Class **Texture Space**

Synopsis A planar mapping into constant **x**.

Description A texture space in which all points in space are mapped onto a plane of constant x. The positive z-axis is taken as an "up" direction.

An overall scaling factor is specified by argument "**scale**". Increasing the value of this factor will have the effect of increasing the scale of all wrapped textures that use this texture space.

"y plane"

Class **Texture Space**

Synopsis A planar mapping into constant **y**.

Description A texture space in which all points in space are mapped onto a plane of constant y. The positive z-axis is taken as an "up" direction.

An overall scaling factor is specified by argument "scale". Increasing the value of this factor will have the effect of increasing the scale of all wrapped textures that use this texture space.

"z plane"

Class **Texture Space**

Synopsis A planar mapping into constant **z**.

Description A texture space in which all points in space are mapped onto a plane of constant z. The positive y-axis is taken as an "up" direction.

An overall scaling factor is specified by argument "scale". Increasing the value of this factor will have the effect of increasing the scale of all wrapped textures that use this texture space.

Texture Placement

In the following paragraphs we will examine some particular cases of texture placement. We will then examine in detail the Interactive Texturing tool which allows to position textures working interactively in the modeling views.

Logo mapping

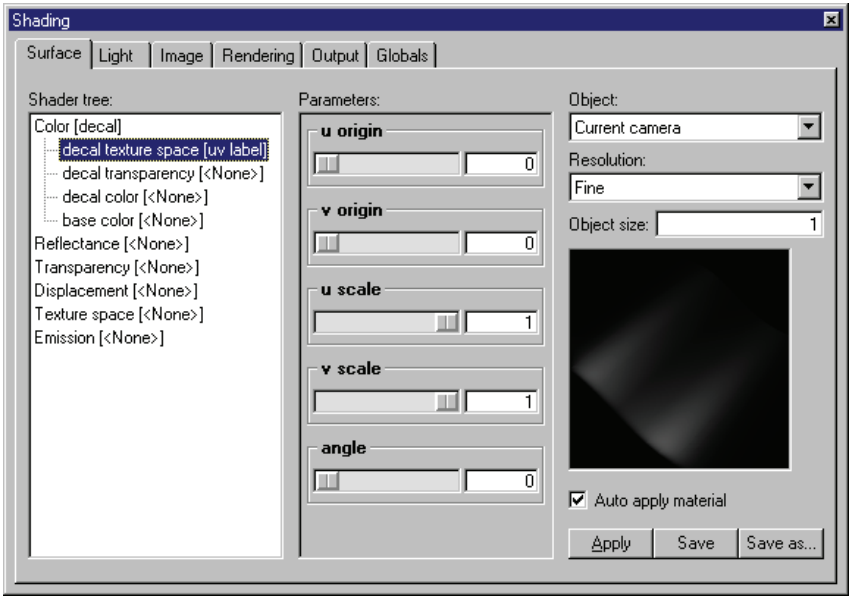
A common case of texture placement is when you need to apply a logo over a surface.

Two textures must be prepared. The first one - *sT-logo* -will be used as the label and it must have a black background (in our example we have created a magenta "sT" logo). The second texture – *sT-mask* - will be used to mask the background of the first. Although the content is the same, the colors will be different: white the background, black the logo.

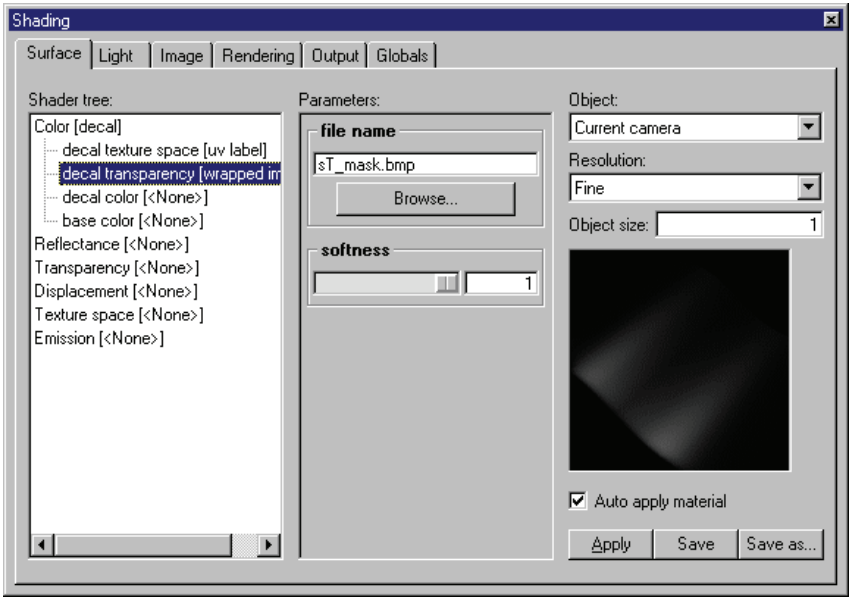
sT-logo

sT-mask

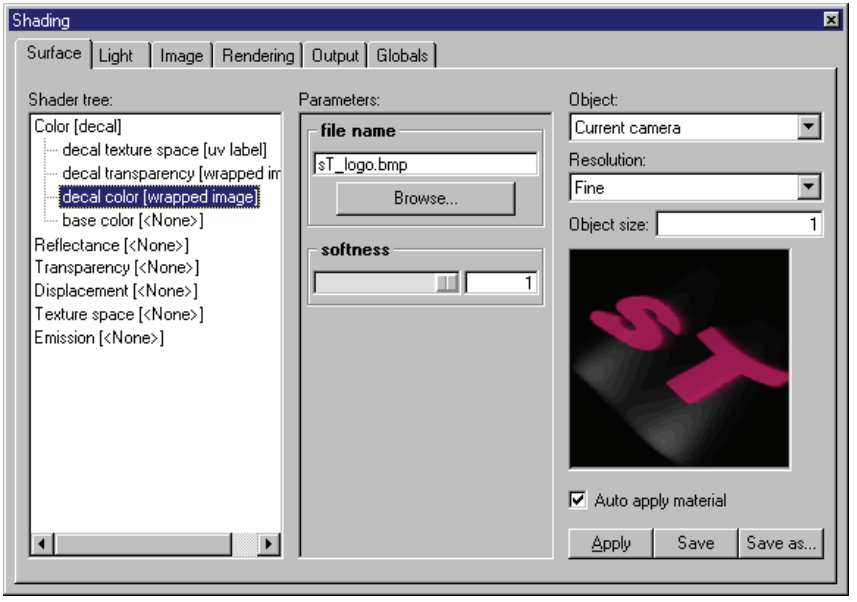
After selecting the surface, open the Shading panel. Choose "**decal**" shader from the "**Color**" type. Then, specify the texture spacing: choose "**uv label**" from the "**decal texture space**" list.



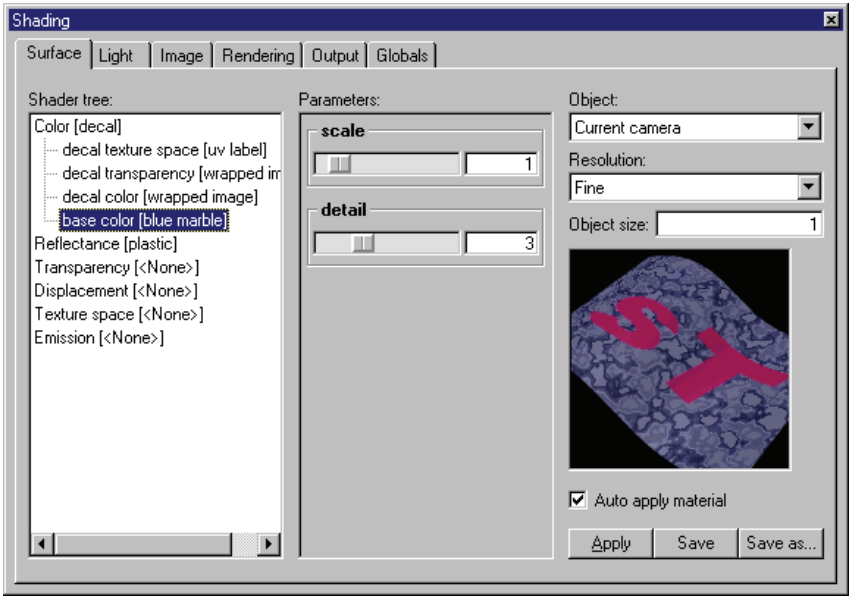
From the "**decal transparency**" list choose "**wrapped image**". Through the "file name" parameter, select the *sT-mask* texture.

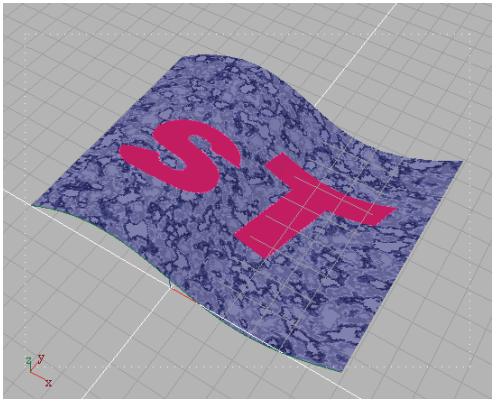


From the "**decal color**" list choose "**wrapped image**" and, through the "file name" parameter, select the *sT-logo* texture.



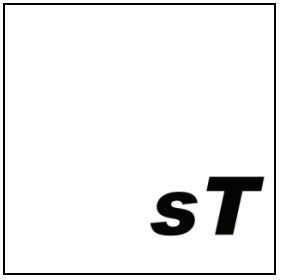
From the "**base color**" list choose "**blue marble**" or any material you like.



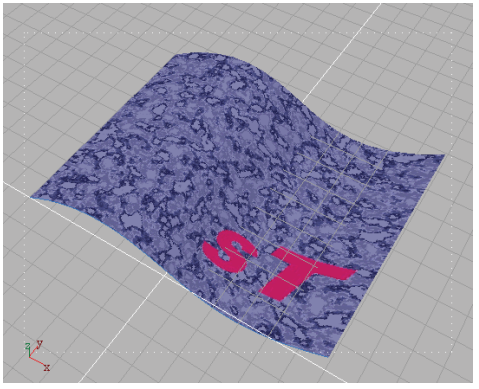


If you want to position the logo in a corner you should prepare the two textures as below.

sT-logo



sT-mask

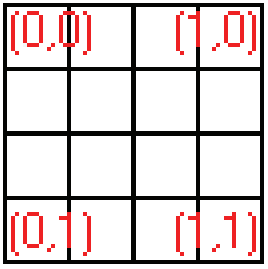


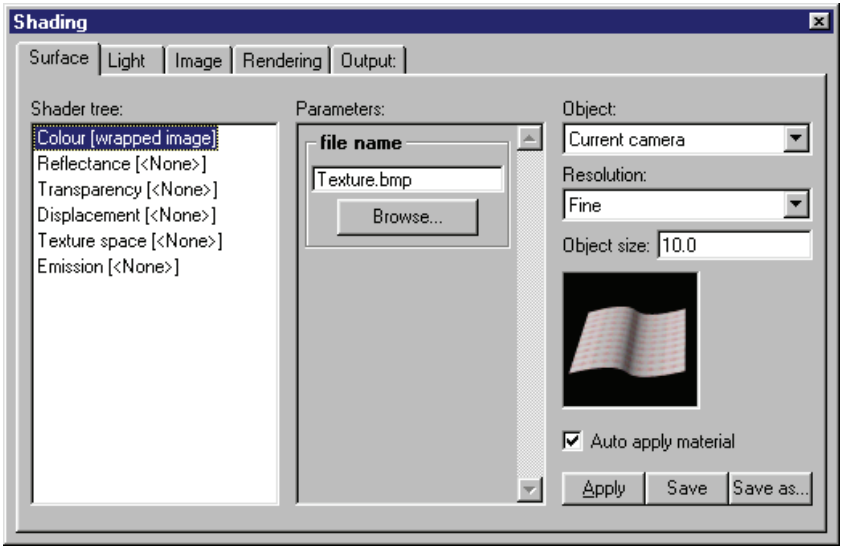
***Note**: In case you want to apply a label without transparency effects - for example, the label of a wine bottle - you do not need to create the masking texture and use the transparency argument.*

U and V repeat

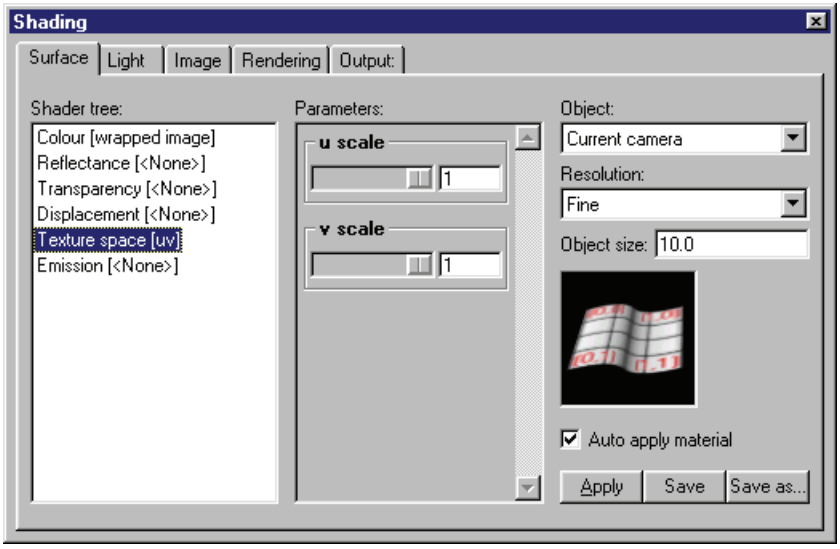
Sometimes you may want to repeat a texture over a surface. Let's see how this is done.

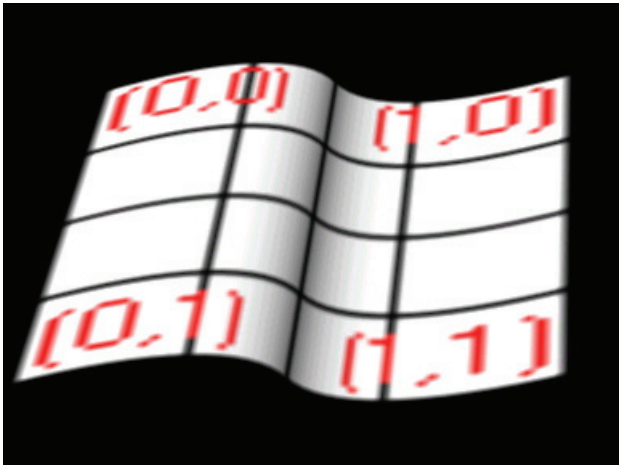
Any texture can be easily applied to a surface by choosing the "wrapped image" shader form the *Color* type.



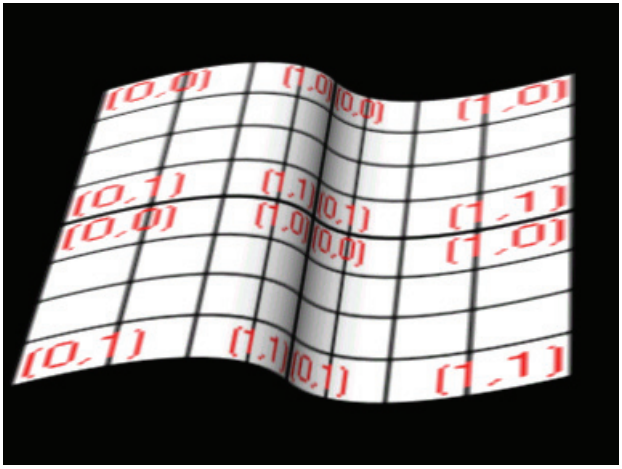


To control the number of repeats along the U and V directions, choose "uv" from the *Texture Space* shaders and adjust the "u scale" and "v scale" parameters.





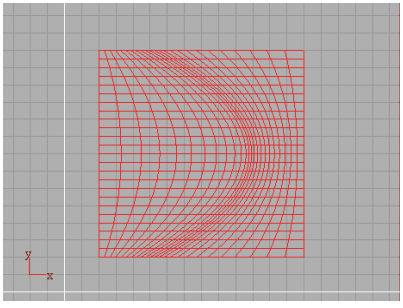
To repeat the texture twice along both U and V, just set “u scale” and “v scale» as in the figure below.

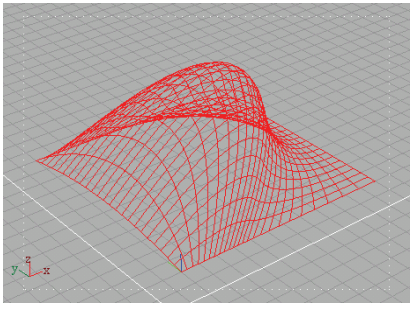


This technique can be used, for example, to create a tiled floor.

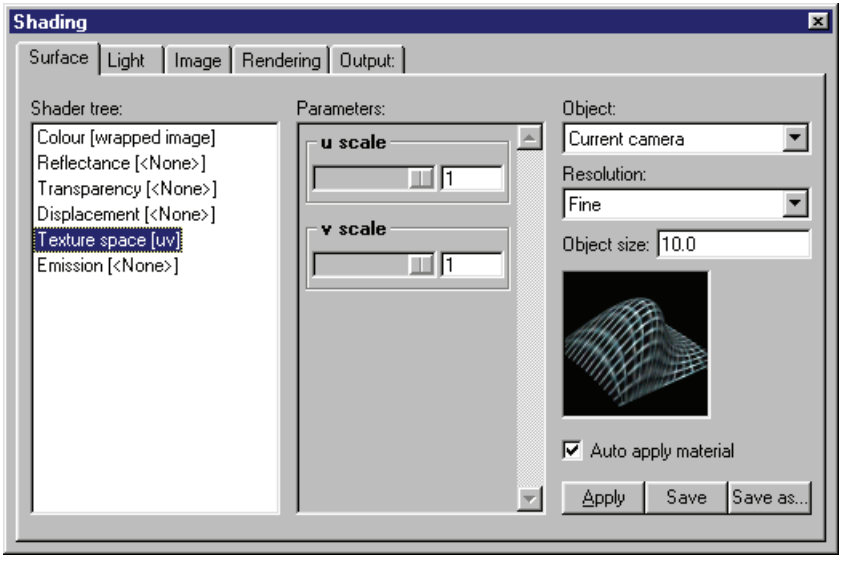
Texturing an uneven surface

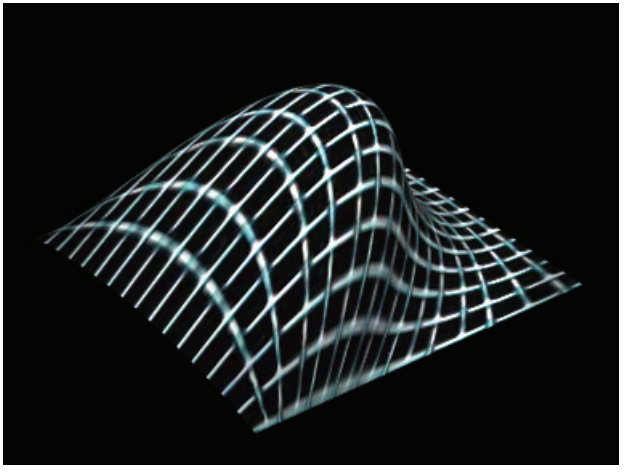
Another interesting case is when you need to place a texture into the UV space of an uneven surface, i.e. a surface that appears to be stretched as in the figures below.





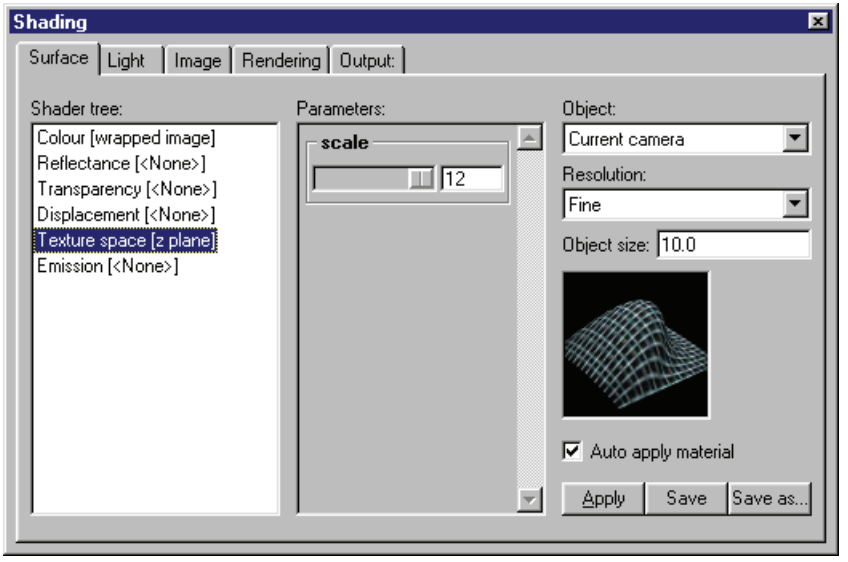
If you use the “uv” Texture Space shaders, the texture will follow the surface patches layout and the result will be an uneven texture.



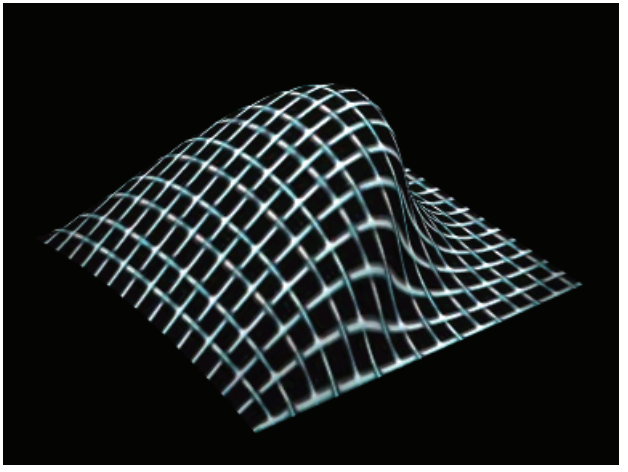


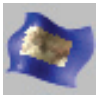
Although this effect could sometimes be exactly what you are looking for, most often you will probably want the texture to be evenly mapped over the surface.

In this case, use the “z plane” shader as shown in the figure.



The texture will appear as if mapped over an even surface.





Interactive Texturing

Interactively controls all parameters of texture mapping shaders. In other words, thanks to this tool you can position textures and more in general any material containing a texture mapping shader. Supported shaders are the most commonly used, i.e. UV, planar and cylindrical.

Please, note that this tool is NOT provided for applying materials (for this purpose you must use the shading panel) but only for positioning materials that you have already assigned.

How to use

1. Select the object whose material you want to position. Note that selected object must already have an assigned material containing texture mapping shaders.

2. Click the **Texture Positioning** icon or choose the **Tools** ▸ **Texturing** ▸ **Texture Positioning** command from the menu.

3. Open the Modeling Tool panel and choose, in the *Active Maps* parameter, the texture mapping shader you want to modify.

4. Set the parameters of the modeling tool to obtain the desired result.

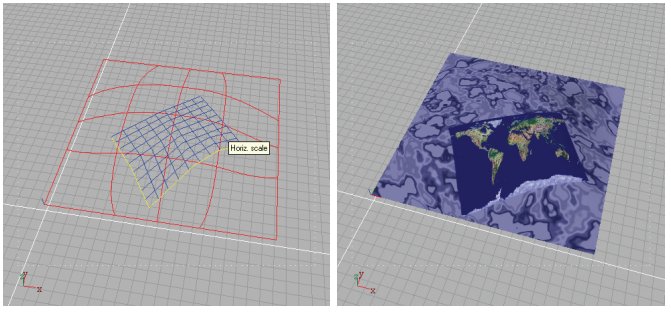
The first parameter of the Modeling Tool panel shows a list of texture mapping shaders active in the selected object. The list can have more than one element in case of decal shaders; in all other cases the list will include only one element.

After selecting a shader in the list, the appropriate mapping tool is activated. The texture to be mapped is represented as a blue grid. The grid is pickable and can be moved to position the texture.

For UV and planar mapping, the grid borders have a different function. Top and right borders allow the user to scale the texture (vertically and horizontally) while the bottom and the left borders (displayed in yellow) can be used to rotate the texture.

In particular, for **UV mapping** the grid lies over the surface to which the texture is applied. The texture can be moved and scaled in the UV domain [0,1]x[0,1].

If the current object is composed of more than one surface, the texture is mapped in all surfaces at the same UV position.

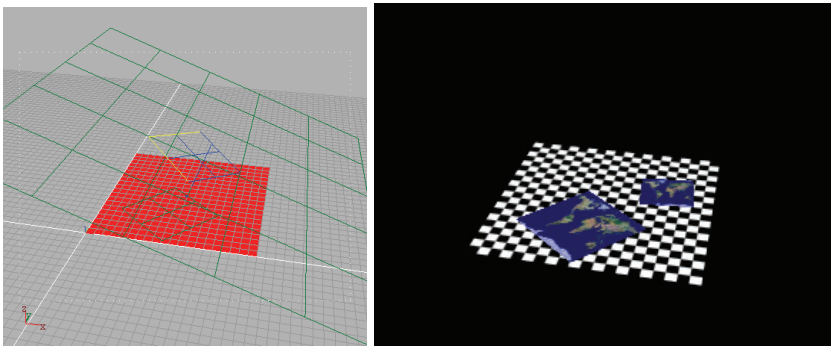


Note: You should pay attention to texture mapping type in polygonal object materials as UV mapping is not supported.

In case of **planar mapping**, the projection plane is shown in green. The blue grid representing the texture lies on this plane and can be moved, rotated and scaled just like UV mapping.

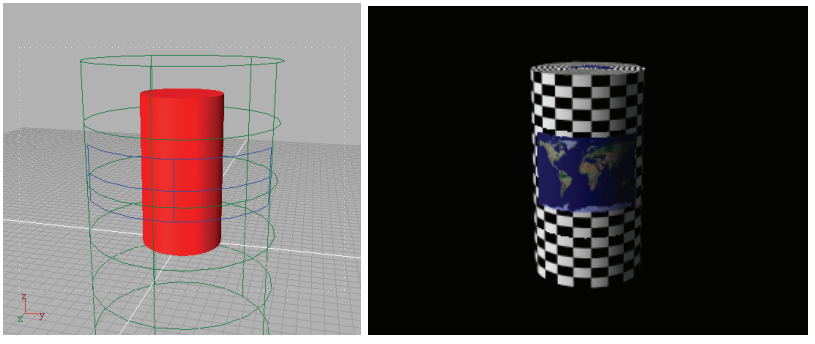
User can also edit the direction of the projection by changing the orientation of the projection plane in space (the projection direction is always perpendicular to the plane).

When you are working with NURBS objects you can also enable the Show projected grid option which is useful to see the exact position of the texture on the objects.



Cylindrical mapping is similar to planar mapping. The difference is that the projection surface shown in green is a cylinder instead of a plane. Moreover, the texture can only be moved and scaled but not rotated (the rotation of the texture on the surface can be achieved by changing the projection direction).

In this case the *Show projected grid* options is not available.



Note: It is possible to modify texture mapping shaders of more than one object simultaneously. Note that texture mapping shaders that can be edited are those present in the material of the first selected object.

Options

Active maps

The texture mapping shaders that can be interactively modified. If selected objects have different materials only shaders applied to the first selected object and present in materials of all other objects can be edited.
Modeling Tool panel. Select from the list.

Mapping Type

The mapping type of the selected texture mapping shader (this parameter is read-only). The supported mapping types are: UV, planar and cylindrical (the only not supported texture space shaders are *auto axis* and *spherical*.

Options

Various options to control user interaction.
Modeling Tool panel. Switch one of the following radio buttons on:
Interactive changes changes interactively the shaders (not only at mouse button up). This is useful (but can be slow) in texture visualization mode.
Show projection surfaces shows projection surfaces (plane or cylinder) when the mapping type is planar or cylindrical.
Show projected grid shows the exact position of the texture when the mapping type is planar.

U Origin

To set the U position of the texture in a surface (active only for the UV mapping).
Modeling Tool panel. Set the position either by moving the slider or by typing it in the data field and pressing Enter.

V Origin

To set the V position of the texture in a surface (active only for the UV mapping).
Modeling Tool panel. Set the position either by moving the slider or by typing it in the data field and pressing Enter.

Angle

To set the angle of the texture with respect to the U isoparametric lines of a surface (active only for the UV mapping).
Modeling Tool panel. Set the position either by moving the slider or by typing it in the data field and pressing Enter.

Default directions

To set the projection direction for planar and cylindrical mapping.
Modeling Tool panel. Switch one of the following radio buttons on:
X, Y, Z: projection direction is aligned to the world X, Y or Z axis.
Horizontal, Vertical: projection direction is horizontal or vertical on the active view.
Custom: user can specify a custom direction both interactively and numerically.

Orbit

The starting point of the projection direction.
Modeling Tool panel. Type the new values in the XYZ data fields and confirm them by pressing Enter.

Center

The end point of the projection direction.
Modeling Tool panel. Type the new values in the XYZ data fields and confirm them by pressing Enter.

Horiz. offset

For planar projection, it is the displacement of the texture along the horizontal direction of the projection plane.
For cylindrical projection, it is the angle in degrees that identifies the position of the texture horizontally around the projection cylinder axis.
This parameter is disabled for UV mappings.
Modeling Tool panel. Set the offset value either by moving the slider or by typing it in the data field and pressing Enter.

Vert. offset

For planar projection, it is the displacement of the texture along the vertical direction of the projection plane.
For cylindrical projection, it is the displacement of the texture along the projection cylinder axis.
This parameter is disabled for UV mappings.
Modeling Tool panel. Set the offset value either by moving the slider or by typing it in the data field and pressing Enter.

Azimuth angle

The angle between the projection direction and the world XY plane.
This parameter is disabled for UV mappings.
Modeling Tool panel. Set the angle either by moving the slider or by typing it in the data field and pressing Enter.

Radial angle

The orientation angle of the projection direction around the world Z axis.
This parameter is disabled for UV mappings.
Modeling Tool panel. Set the angle either by moving the slider or by typing it in the data field and pressing Enter.

Inclination

The inclination of the texture respect to the horizontal direction of the projection plane (active only for planar mapping).
Modeling Tool panel. Set the inclination value either by moving the slider or by typing it in the data field and pressing Enter.

Horiz. scale

For UV mapping, it is the U extension (parametric) of the texture (in the domain [0..1]).
For Planar mapping, it is the linear horizontal extension of the texture to be projected.
For Cylindrical mapping, it is the angular extension of the texture around the axis of the projection cylinder (expressed as a percentage of 360° as in the shading panel).
Modeling Tool panel. Set the scale value either by moving the slider or by typing it in the data field and pressing Enter.

Vert. scale

For UV mapping, it is the V extension (parametric) of the texture (in the domain [0,1]).
For Planar mapping, it is the linear vertical extension of the texture to be projected.
For Cylindrical mapping, it is the linear vertical extension of the texture to be projected along the axis of the projection cylinder.
Modeling Tool panel. Set the scale value either by moving the slider or by typing it in the data field and pressing Enter.

Center

UV mapping. To make the center of the texture coincident with the center of the parametric domain of the surfaces (at [0.5, 0.50]).
Planar mapping, **Cylindrical mapping**. To make the center of the texture coincident with the center of the projection plane (or cylinder) and the center of projection coincident with the center of bounding box of the selected objects.
Modeling Tool panel. Press the **Center** button .

Fit

UV mapping. To make the texture cover exactly the entire surface.
Planar mapping. To make the projected texture optimally cover all selected objects.
Cylindrical mapping. No effect.
Modeling Tool panel. Press the **Fit** button.

Horiz. grid detail

Number of horizontal subdivisions of the grid representing the texture.
Modeling Tool panel. Set the scale value either by moving the slider or by typing it in the data field and pressing Enter.

Vert. grid detail

Number of vertical subdivisions of the grid representing the texture.
Modeling Tool panel. Set the scale value either by moving the slider or by typing it in the data field and pressing Enter.

5.4 Lights

In real world environments light sources affect the appearance of objects and define their shapes. ***solidThinking*** allows you to reproduce these effects. Just as in a photographic set, you can position and control an unlimited number of light sources.

solidThinking handles light sources by treating them as shaders, which have positions, intensities, and other properties, such as directions, colors, etc. And just like any other shader, each light source has parameters that can be edited.

During the shading of an object, light source shaders are accessed to find the intensity, color, and angle of incidence of light from each source, which, along with the color, texture, reflectance component, etc., of the object, can be used to establish its appearance at a particular point.

This lighting method offers you enormous advantages compared to any other method, giving you the flexibility to modify illumination of the scene very quickly and precisely.

In ***solidThinking***, light sources are not modeled in a physically-based manner. For example, intensities are given by using arbitrary units that do not directly relate to the power output of a lamp, or any other such physical measurement. A physically accurate simulation of lighting conditions can be achieved when using the Radiosity Processor Module: in fact, the role of the shaded rendering engine is intended only to generate shaded images that look realistic.

By default the scene is illuminated by a single eye light.

Light Source types

solidThinking provides several standard light source prototypes, namely ambient, distant, eye, goniometric, point, projector, and spot lights. These light source types should provide for adequate simulation of real world lighting phenomena. They can be shortly described as follows:

Ambient

A non-directional light providing uniform illumination.

Point

A light from a point source. Examples of a point light are bulbs and candles.

Spot

A directional light from a spot source with a cone of illumination

Distant

A directional light from a distant source. It is made up of parallel rays of light, all of which follow the same direction. As the result, surfaces with the same orientation receive the same quantity of light. The sun is considered a distant light because it is so far from us that rays arriving to earth are parallel.

Sun

A light source designed to mimic sunlight.

Sky

Sky light source.

Simple Sky

A fast approximation to a true sky light.

Environment

Light source which provides lighting effects from the current environment (by treating the environment texture as a light source). For best results a high dynamic range image should be used as the environment texture.

Simple Environment

An alternative environment light source which may in some cases be slightly quicker than the "environment" light, at the cost of reduced quality.

Projector

Projects a supplied image onto illuminated objects.

Goniometric

Light source for which the distribution of light is specified in using industry-standard data.

In addition to the above light source types, there are **Area Light sources**, which are only usable within the Radiosity Processor Module and are explained in the Radiosity paragraph:

The next paragraphs will examine in detail how to add any light to the scene by using the Light tool and assigning one of the different light shaders available.

New Light

It allows you to add and position any light source in the 3D scene.

How to use

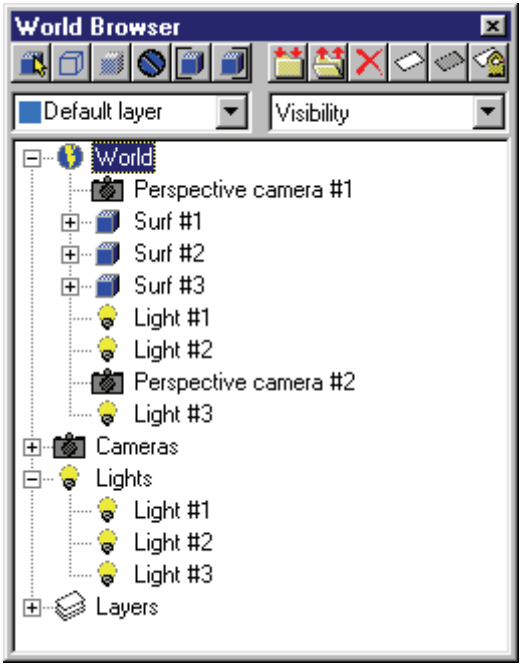
1. Click the **Light tool** icon or choose the **Tools ▸ Cam/Light ▸ Light** command from the menu.

2. The program console prompts **Local axes origin**.

3. Locate the curve axes origin following one of the two methods:
   a) Click and drag in any view and release the mouse button to confirm placement.
   b) Type its XYZ coordinates in the console (values must be separated by a comma, i.e. **1,2,3**) and press Enter to confirm. If you press Enter without typing any value in the console, the axes origin of the curve will be placed in correspondence with the XYZ origin.

At this point you have positioned a light source in the 3D scene but no lighting characteristic has been defined yet. Now, any light shader can be assigned.

To successively select a light, you can simply click it in any view or choose it using the Hierarchy Browser, which lists all the lights in the 3D scene twice. Look at the following picture.



The first time, under World, they are displayed in order of insertion in the scene; the second time, under Lights, they are displayed together so that you can have a quick feedback on which lights are in the scene.

Options

Position

To modify XYZ coordinates of the light position.
Views. Choose the **Edit** mode. Pick the *Position* hotSpot and drag it. Release the mouse button when you have reached the desired location.
Modeling Tool panel. Type the coordinates values in the XYZ data fields and confirm them by pressing Enter.

Target

To modify XYZ coordinates of the light target.
Views. Choose the **Edit** mode. Pick the *Target* hotSpot and drag it. Release the mouse button when you have reached the desired location.
Modeling Tool panel. Type the coordinates values in the XYZ data fields and confirm them by pressing Enter.

Aperture

To adjust the angle of distribution of the light.
Modeling Tool panel. Set the degrees value either by moving the slider or by typing it in the data field and pressing Enter.

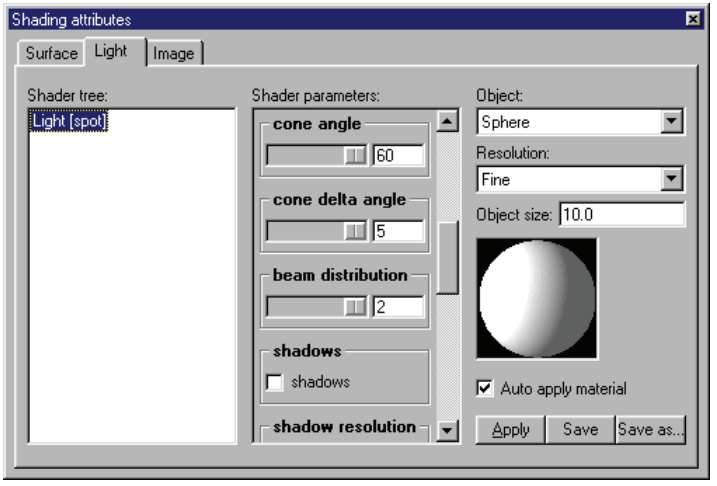
***Note**: Depending on the light shader you assign to the Light tool, the above mentioned options may be accessible or not. Specific light parameters are available as arguments of each light shader and listed in the Shading panel.*

Assigning a Light Shader

Once you have positioned a light in the 3D scene using the light tool, you are ready to assign a light shader.

Select the light. Open the **Shading** panel (choose the **Managers ▸ Shading** command or use the Ctrl + 3 shortcut). This panel displays three tabs (Surface, Light, and Image) on the upper left corner.

Select the **Light** tab. The **Shader tree** column on the left lists only the **Light** shader type.



The name of the shader assigned is displayed in [square brackets]. <None> indicates that no shader is currently assigned.

To assign a shader just press the right mouse button when you are over the Light shader class: a pop-up menu will be displayed allowing you to choose among the available shaders by simply dragging.

When you are over the shader you want to use, release the mouse button: the name of the chosen shader will be displayed, instead of <None>, in square brackets.

In the middle of the panel, the **Parameters** column lists the shader arguments by means of which you can customize the shader and define its lighting characteristics.

Depending on the number of arguments a shader has, the list can be scrollable or not.

Saving light settings

solidThinking allows you to save any light setting and store it in the extensible **Materials library** so that it can be quickly retrieved for any successive use.

To save a new light setting, press the **Save** button on the bottom right of the Shading panel.

The **Save as Material** panel is displayed. Choose the **Group** where you want to store the new light setting by scrolling the pop-up list, for example Lights. If you want to create a new group, simply type a different name not included in the pop-up list. Then, type the name and click **OK** to confirm.

You will find the new light setting in the Materials library under the group you chose.

Editing light settings

Any light setting previously stored in the Materials library may be modified at any time by changing argument values.

After editing a light setting assigned to a light tool, you have two options:

a) To overwrite the original light setting and save changes under the same name, simply press the **Save** button.
b) To retain the original light setting and create a new one under a different name press the **Save as…** button. The **Save as Material** panel is displayed. Choose the **Group** where you want to store the new light setting by scrolling the pop-up list, type the name and click **OK** to confirm.

Shadows

One feature, which applies to light source shaders but not to shaders of other classes, is the specification, creation and manipulation of shadows.

Of the six light sources supplied, distant, point, projector and spot light sources are able to cast shadows. Ambient and eye light sources do not cast shadows.

solidThinking supports shadowing in two different methods: shadow mapping and shadow ray casting. Note, however, that the shadow ray casting technique is only used by the ray tracer renderer.

The shadow ray casting works by following rays from a point on an object's surface toward light sources. Thus, it establishes whether or not there is a direct line of sight from the point to a given light source, and if it is lit by, or in shadow from, the source.

The shadow mapping technique involves shadow information for each light source that is able to cast shadows. This information may then be applied in a view-independent manner to subsequent images that are rendered. The properties of the shadows are specified by means of some of the arguments to the corresponding light source instance. Before a light source is able to cast shadows using this method, the preprocessing of the shadow data must be undertaken at the explicit direction of the application program. If any changes are made to the specification of the light source that affect the shadows, then the preprocessing of shadow data must be repeated.

The ray casting technique can only produce hard-edged shadows, which tend to detract from the realism of an image. On the contrary, the shadow mapping technique is capable of producing shadows with softened edges.

The technique that is applied is determined from the ‘resolution’ value of the light shader. The resolution of shadowing is set for each light source individually using the shader argument "shadow resolution". If this value is set to zero for a light source, then shadowing from that source will be calculated using ray casting, producing hard-edged shadows.

When light passes through objects that are semi-transparent, you expect light to be partially occluded. For example, sunlight passing through a stained-glass window can produce a multi-colored ‘shadow’ on the floor.

solidThinking can reproduce this sort of effect, when shadows are produced using ray-casting (i.e., the light sources involved must be declared to produce ray-cast shadows by setting the "shadow resolution" shader argument to zero). The shadow-mapping technique is not adequate to provide this effect, and it will produce shadows as if the object were completely opaque.

When using ray-casting, a surface rendered using a "plain" colored transparency will cast shadows that are changed in hue as if the light had travelled through a colored medium.

On the other hand, a surface with many holes (as would be the case if the "eroded" transparency shader were used) will result in a mottled shadow being cast.

A "wrapped image" transparency shader will result in the image being used to cast shadows. This is how the effect of stained-glass, or windows with frames within the main shape, can be achieved without complicated modeling.

While it is possible to create some very interesting effects using this technique, keep in mind that it is very computationally intensive.

Controlling Shadow Production for Each Object

The production or otherwise of shadows can be controlled for each object in the scene, as well as for each light.

By default, all objects will cast and receive shadows, when illuminated by lights which cast them. However, there may be situations where this is undesirable, for instance, certain objects cannot produce a visible shadow, due to the viewing position: excluding these objects from shadow calculations can result in improved rendering performance; both in terms of increased speed and reduced memory usage.

This can be performed through the hierarchy browser (open it using the **Managers ▸ Browser** command). By right clicking one or more selected objects in the browser, a popup menu appears allowing you to instruct the program as whether an object must **Cast shadows** or not and whether it must **Receive shadows** or not.

You can resort to these options also to illuminate certain areas of a scene without creating shadows, without the need for a complex system of 'fill' lights as would be used in a theatre or movie set.

Light Shaders

Let's now examine in detail shaders of the Light class.

For each shader we are providing synopsis, arguments, related default values, and description.

You can gain confidence with each shader by modifying parameters and verifying how this affects its lighting characteristics.

"ambient"

Class **Light Source**

Synopsis Uniform illumination.

Description A non-directional light source that equally illuminates all surfaces regardless of orientation.

The intensity of the source (as a scalar quantity) and its color are specified as arguments "**intensity**" and "**color**" respectively.

Ambient light is often used as a secondary light source, supporting a stronger directional light type (distant, spot).

"distant"

Class **Light Source**

Synopsis Parallel light from a distant source.

Description A directional light source emitting light parallel to a given direction vector as if coming from a very distant source, such as the sun.

The direction of light is specified by means of two arguments: "**location**" and "**to**", such that the direction vector is taken as being from the "location" point towards the "to" point.

The intensity of the light (as a scalar quantity) and its color are specified by means of the arguments "**intensity**" and "**color**" respectively.

Shadows may be turned on by checking the "**shadows**" argument on. The resolution of the shadow map that is used in the preprocessing of shadows is specified in argument "**shadow resolution**". If this is given as 0, then ray casting will be used to calculate the shadow, which will result in a hard-edged shadow.

The quality of the shadows is specified by "**shadow quality**": a value of 1 corresponds to low quality, while a larger value such as 9 gives higher quality.

A value that determines how soft the boundaries of shadows appear is specified in argument "**shadow softness**". A value of 1.0 gives rise to hard shadow boundaries, and a larger value such as 4.0 causes soft boundaries with penumbrae.

Distant light sources may be used in radiosity simulations, in which case only the "location", "to" and "radiant" power arguments will be relevant.

Since the distant source is thought of as a light source at infinity, it is not meaningful to define the power it emits. Instead, we define the effect the source has on objects in the scene. The "**power**", defined separately for the red, green and blue wavelengths, defines the radiant energy landing on a planar region, of unit area, which is facing directly towards the light source. The power should be specified in units of Watts.

"goniometric"

Class **Light Source**

Synopsis Light source with intensity distribution specified in an industry standard fashion.
For use with both radiosity and shaded rendering.

Description A goniometric light source can emit widely varying amounts of light energy in different directions. One goniometric source could behave exactly like a point light source; another could behave exactly like a spot light source, and a third could look nothing like either of those.
A goniometric light gets its intensity distribution function (how much light goes in any one direction) from an industry-standard file.

This file can be in any one of three formats; the shader will work out the type you have provided. The three formats supported are as follows:
- *CIE* (Commission Internationale de L'Eclairage). An international standard. Detail of the file format may be found in the CIE Technical Report CIE-102, 1993, Recommended File Format for Electronic Transfer of Luminaire Photometric Data', ISBN 3-900-734-40-2.
- *IESNA* (Illuminating Engineering Society of North America). A North American standard. Details can be found in the technical report IESNALM-63-95, 'IESNA Standard File Format for Electronic Transfer of Photometric Data'.
- *CIBSE* (Chartered Institution of Building Services Engineers). A British standard. Details can be found in CIBSE Technical Memorandum 14 (TM14, 1988), 'CIBSE Standard File Format for the Electronic Transfer of Luminaire Photometric Data.

All of these formats are formatted ASCII text files (`flat files') containing various keywords to identify the data which follows.

The meanings of the shading arguments are as follows:

"**from**". The location of the light, in world coordinates.

"**to**". The point that defines where the "South Pole" of the (polar) intensity distribution should be. Again, world coordinates.

"**equator zero**". The point that defines a reference point on the "equator" of the (polar) intensity distribution.

"**shadows**". A boolean that specifies whether or not the light should cast shadows.

"**file name**". A file specifying, via a spherical polar distribution, how much light is emitted in different directions.

"**power units**" (only with Radiosity). This is used to specify the power in photometric units ("lumens") or in radiometric units ("watts").

"**power**" (only with Radiosity). This is a single float value that defines the total power output of the source. If "power units" is set to "lumens", then this is the luminous power of the source. If "power units" is set to "watts", then this is the sum of the red, green and blue components of the energy (in Watts) leaving the source. In this latter case the *red: green: blue* ratio is determined by the color arguments.

"**color**" The RGB color of the emitted light. If used in conjunction with color temperature, this will be combined with whatever RGB color comes from that.

"**color temperature**" A temperature, in Kelvin, which should be greater than 1000K if it is not to be ignored. If set to 0, or never set, then the color argument alone defines the color of emitted light.

"**lamp type**". This is a string interface to the "color temperature" argument. One has the option of specifying one of nine different strings, each representing a common lamp type. These strings map directly onto a color temperature, which is then used in color determination. The table below shows the color temperatures of some common lamp types:

Lamp Type	*Color Temp (K)*
warm white fluorescent	3020
white fluorescent	3450
cool white fluorescent	4250
daylight fluorescent	6250
clear mercury	5710
improved color mercury	4430
low pressure sodium	1740
high pressure sodium	2100
tungsten halogen	3190

"point"

Class **Light Source**

Synopsis Light from a point source.

Description A light source emitting light from a point equally in all directions.

The position of light is specified by means of argument "**location**".

Intensity (as a scalar quantity) and color are specified by means of the arguments "**intensity**" and "**color**" respectively.

An argument "**fall off**" is provided and it may be used to indicate how the intensity of light emanating from the point varies with distance from the point itself. It can either be *constant*, *inverse*, *inverse square*, *unclamped inverse*, or *unclamped inverse square*. If fall off is set to *constant*, the light intensity does not vary with distance from the source. If fall off is set to *inverse*, the light intensity varies with the inverse of the distance from the source.

If fall off is set to *inverse square*, the light intensity varies with the inverse squared distance from the source.

Shadows may be turned on by checking the "**shadows**" argument on. The resolution of the shadow map that is used in the preprocessing of shadows is specified by "**shadow resolution**". If this is given as 0, then ray casting will be used to calculate the shadow, resulting in a hard-edged shadow.

The quality of shadows is specified by "**shadow quality**": a value of 1 corresponds to low quality, and a larger value such as 9 gives higher quality.

A value that determines how soft the boundaries of shadows appear is specified in argument "**shadow softness**". A value of 1.0 gives rise to hard shadow boundaries, and a larger value such as 4.0 causes soft boundaries with penumbrae.

"**power**" specifies the power output of visible light energy by the source. This value is given independently for the red, green and blue components. The power should be specified in units of Watts.

"**color temperature**" defines a temperature, in Kelvin, which should be greater than 150K if it is not to be ignored. If set to 0, or never set, then the color argument alone defines the color of emitted light.

You can define whether the luminous intensity has been supplied in *candela* or in *kilocandela* in the "**intensity units**" field. The latter has the advantage that intensity values around 1:0 are representative of lights that one might well find in the modern built environment.

The "**scattering**" option must be enabled if you want to use the “light scattering” foreground shader in order to obtain volumetric shadows.

"projector"

Class **Light Source**

Synopsis Light source that projects a focused image, rather than plain light.

Description A light source that projects a focused image, rather than plain light, simulating the effect of a slide projector.

It also allows the effect of light passing through a stained glass window to be modeled, if a smaller angle of illumination is used.

A black and white image allows light to be ‘stencilled’, which for example can give the effect of a window with shadows cast by the window frame.

A "projector" light source behaves similarly to a "spot" light source, and is defined in the same way, except that an image file is also specified as one of its parameters by the "**file name**" argument.

The light position is specified by means of argument "**location**"; the direction of the axis of the cone points from this position to the point specified by argument "to".

The argument "**up vector**" defines the orientation of the image in world-space. A sensible value for this argument ensures that the image is `the right way up'.

Intensity (as a scalar quantity) and color are detailed in arguments "**intensity**" and "**color**" respectively.

The light color will, of course, be filtered by the image; a non-white value for "**color**" will be rather like using a colored light-bulb in a slide projector.

The argument "**fall off**" may be used to indicate how the intensity of light emanating from the point varies with distance from the point. It can either be *constant*, *inverse*, *inverse square*, *unclamped inverse*, or *unclamped inverse square*. If fall off is set to *constant*, the light intensity does not vary with distance from the source. If fall off is set to *inverse*, the light intensity varies with the inverse of the distance from the source. If fall off is set to *inverse square*, the light intensity varies with the inverse squared distance from the source.

The angle of the frustum within which light is emitted from the light source is specified in degrees by the value passed as argument "**angle**".

Shadows may be turned on by checking the "**shadows**" argument on. "**shadow resolution**" lets you set the resolution of the shadow map that is used in the preprocessing of shadows. If this is given as 0, then ray casting will be used to calculate the shadow, resulting in a hard-edged shadow.

The quality of the shadows is specified by "**shadow quality**": a value of 1 corresponds to low quality, while a larger value such as 9 gives higher quality.

"simple sky"

Class **Light Source**

Synopsis A fast approximation to a true sky light.

Description The “simple sky” light shader is simpler to use, and usually faster to execute, than the "sky" shader, but is still capable of providing good results.

The "simple sky" light models the sky as a simple, uniform dome, rather than attempting to model the non-uniform distribution of light across a real sky.

The shader has arguments for "intensity", "intensity units", "color", "color temperature", "shadows", "shadow type", "shadow resolution", "shadow quality", "shadow softness", and "shadow tolerance" which are all pretty standard for light source shaders.

The arguments which are specific to the "simple sky" light source shader are as follows:

"up"**:** The "up" vector defines the orientation of the sky hemisphere with respect to the scene. Specifically, it is a vector that joins any point in your scene to the zenith (highest point) of the sky hemisphere.

"noise factor"**:** This control has no effect if soft shadows are requested, but when hard shadows are being calculated the sample directions are varied (‘jittered’) slightly in an effort to replace sampling artifacts with noise. Unlike the "sky" shader, every sample will be jittered if the "noise factor" is set to a large enough value. Values can vary from 0.0 (no jittering) to 1.0 (maximum jittering).

"sky"

Class **Light Source**

Synopsis Simulates illumination by the sky hemisphere.

Description A light source to account for the illumination from the sky (but not the direct contribution due to the sun itself). The sky is modeled as a hemisphere with an infinite radius, positioned above the centre of the current scene.

The orientation of the sky is specified by the "**north**" and "**up**" shader arguments; the latter defines the horizon for every point being illuminated. The position of the sun is specified via the "**sun altitude**" and "**sun azimuth**" arguments. Whilst the actual contribution of the sun is *not* modelled by the shader, its location will determine the appearance of the sky hemisphere, whose contribution *is* modelled by the shader. The "sun altitude" is the angle, in degrees, of the sun above the horizon. The "sun azimuth" is the angle, in degrees, of the sun around the horizon. An azimuth of 0 means that the sun in the North (defined by the "north" shader parameter); an azimuth of 90 puts the sun in the East, and an azimuth of 270 puts the sun in the West.

The "**type**" argument specifies how the illumination is distributed over the sky. Six standard models are provided; three are from the *Commission Internationale de L'Éclairage* (CIE), and three are from the compatible North American body, the *Illuminating Engineering Society of North America* (IESNA).

Luminance distributions supplied are:

Standard CIE Overcast Sky
Standard CIE Clear Sky
CIE Intermediate Sky
Standard IESNA Overcast Sky
Standard IESNA Clear Sky
Standard IESNA Intermediate Sky

Essentially, the `clear' skies will model the sky as if it were a sunny, cloudless, day (the brightest part of the sky surrounds the sun location) whilst the `overcast' skies model the sky on a cloudy day (where the brightest part of the sky is straight up). An `intermediate' sky models a sky that is neither clear nor cloudy.

The more complex the sky illumination (`clear' is most complex, `overcast' is least complex) the more expensive (compute-power) the light source shader.

To specify the brightness of the light emitted by the sky, use the "**intensity**" argument. This gives the luminance at the sky's zenith.

The "**shadows**" parameter specifies whether or not the sky will cast shadows on the scene.

"spot"

Class **Light Source**

Synopsis Light from a spot source constrained to a cone.

Description A light source emitting light from a single point and constrained to a cone.

The position of the light is specified by means of argument "**location**", and the direction of the axis of the cone points from this position to the point specified by argument "**to**".

The intensity of the light (as a scalar quantity) and its color are specified by means of the arguments "**intensity**" and "**color**" respectively.

The argument "**fall off**" is provided which may be used to indicate the way in which the intensity of light emanating from the point varies with distance from the point. It can either be *constant*, *inverse*, *inverse square*, *unclamped inverse*, or *unclamped inverse square*. If fall off is set to *constant*, the light intensity does not vary with distance from the source. If fall off is set to *inverse*, the light intensity varies with the inverse of the distance from the source. If fall off is set to *inverse square*, the light intensity varies with the inverse squared distance from the source.

The angle of the cone within which light is emitted from the light source is specified in degrees by the value passed as argument "**cone angle**" (a full angle), while the angular distance over which the intensity diminishes is specified by argument "**cone delta angle**" (a half angle).

This angle is measured from "cone angle" and towards the axis, and is specified such that the intensity is varied from the computed value at the angle 0.5 × "cone angle" - "cone delta angle" to zero at angle 0.5 × "cone angle".

The variation of the intensity of light from the axis of the cone outwards may be set with the argument "**beam distribution**". A value of 0.0 for this argument causes uniform intensity across the interior of the beam, and the higher the value, the faster the intensity diminishes with the angle.

Shadows may be turned on by checking the "**shadows**" argument on. The resolution of the shadow map that is used in the preprocessing of shadows is passed in argument "**shadow resolution**". If this is given as 0, then ray casting will be used to calculate the shadow, resulting in a hard-edged shadow.

The quality of shadows is specified by "**shadow quality**": a value of 1 corresponds to low quality, while a larger value such as 9 gives higher quality.

A value that determines how soft the boundaries of shadows appear is specified in argument "**shadow softness**". A value of 1.0 gives rise to hard shadow boundaries, and a larger value such as 4.0 causes soft boundaries with penumbrae.

"**color temperature**" defines a temperature, in Kelvin, which should be greater than 150K if it is not to be ignored. If set to 0, or never set, then the color argument alone defines the color of emitted light.

You can define whether the luminous intensity has been supplied in *candela* or in *kilocandela* in the "**intensity units**" field. The latter has the advantage that intensity values around 1:0 are representative of lights that one might well find in the modern built environment.

The "**scattering**" option must be enabled if you want to use the “light scattering” Foreground shader in order to obtain volumetric shadows.

"sun"

Class **Light Source**

Synopsis A light source emitting light from the Sun (i.e., a very distant source).

Description The light emitted is parallel to the vector between the coordinate system origin and a point which is located "**altitude**" degrees above the horizon in the direction "**azimuth**".

The coordinate system in which "altitude" and "azimuth" are defined is specified by means of "**north**" and "**up**" vectors. Values of "azimuth" of 0, 90, 180 and 270 degrees correspond to North, East, South and West respectively. A value of 90 degrees for the "altitude" argument corresponds to the sun in zenith.

The color of sun is determined by "**color**" and "**color temperature**" arguments. The intensity of the sunlight is controlled by the mean of the "**intensity**" parameter. The intensity can be specified in different intensity units by setting "**intensity units**" to one of the following values: Empirical, Lux, Kilolux, Footcandle.

Parameters "**latitude**" and "**longitude**" define location of scene. Their values are specified in degrees. For "latitude" positive values define position on the north hemisphere, negative values on the south hemisphere. For "longitude" positive values define position on the east hemisphere, negative values on the west hemisphere.

Shadows may be turned on for the sun by means of the flag "**shadows**". The quality of the shadows is specified by "**shadow quality**": a value of 1 corresponds to low quality, and a larger value such as 9 gives higher quality. A value that determines how soft the boundaries of shadows appear is specified in argument "**shadow softness**". A value of 1.0 gives rise to hard shadow boundaries, and a larger value such as 4.0 causes soft boundaries with penumbrae.

5.5 The Camera

The camera is the position from which the 3D scene is viewed.

Any 3D camera is described by its position, target, rotation, and field of view. All these parameters can be modified at any time.

By default, any 3D scene is created with one perspective camera, but you can produce any number of perspective cameras that will be identified by a progressive numbering, i.e. perspective camera #1, perspective camera #2, etc.

The perspective view displays what the active Camera sees.

Creating a new camera is a very easy task, as you will see in the next paragraph.

New Camera

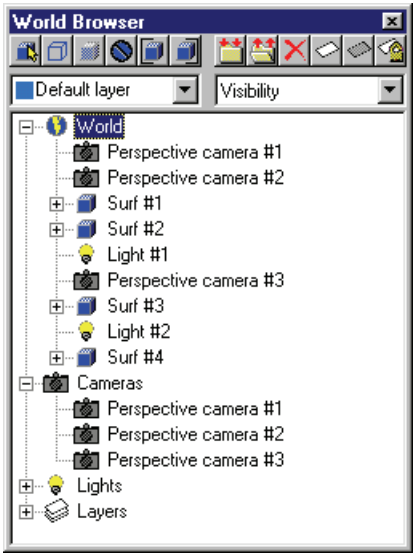
Permits to create and position a new perspective camera in the 3D scene.

How to use

1. Click the **Perspective camera** icon or choose the **Tools** ▸ **Cam/Light** ▸ **Perspective Camera** command from the menu.

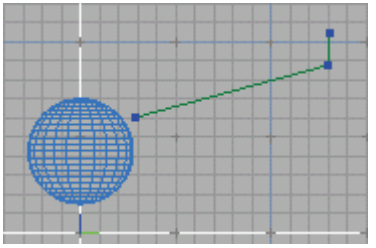
2. A new Perspective camera is added to the 3D scene.

XYZ coordinates of the new camera, when created, are 0,-1, 0 for position and 0,0,0 for target. You can easily adjust them by picking and dragging their hotSpots or by typing new coordinates in the Modeling Tool panel. To select perspective cameras, you must use the Hierarchy Browser, which lists all the cameras in the 3D scene.

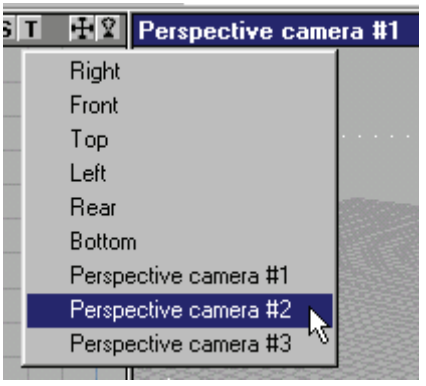


As shown in the previous picture, cameras in the 3D scene, just as lights, are listed twice: the first time, under World, they are displayed in order of insertion in the scene; the second time, under Cameras, they are displayed together so that you can have a quick feedback as to which cameras are in the scene.

Once you select a perspective camera, it will be displayed in the views allowing you to interactively adjust both its position and target.



If you click the view name on the left of the title bar, you will find any defined perspective camera.



The perspective view will display what the Perspective camera you select sees.

Options

Position

To modify XYZ coordinates of the camera position.
Views. Choose the **Edit** mode. Pick the *Position* hotSpot and drag it. Release the mouse button when you have reached the desired location.
Modeling Tool panel. Type the coordinates values in the XYZ data fields and confirm them by pressing Enter.

Look at

To modify XYZ coordinates of the camera target.
Views. Choose the **Edit** mode. Pick the *Look at* hotSpot and drag it. Release the mouse button when you have reached the desired location.
Modeling Tool panel. Type the coordinates values in the XYZ data fields and confirm them by pressing Enter.

Up

To modify XYZ coordinates of the up vector end point.
Views. Choose the **Edit** mode. Pick the *Up* hotSpot and drag it. Release the mouse button when you have reached the desired location.
Modeling Tool panel. Type the coordinates values in the XYZ data fields and confirm them by pressing Enter.

Focal length

To modify the focal length.
Modeling Tool panel. Set the value either by moving the slider or by typing it in the data field and pressing Enter.

5.6 Backgrounds and Foregrounds

Backgrounds and Foreground shaders can be used to perform simple image-manipulation operations.

In particular, background shaders determine the color of pixels behind the geometry in the scene, while foreground shaders determine the relationship between the color of the visible surface at a pixel and the color that is actually displayed at that pixel.

When a shaded image is rendered, often not all pixels of the resulting image are covered in whole or in part by models you have created. The shading of these pixels is governed by the current background shader. A background shader calculates its shade based upon pixel location, and is therefore independent of the view specification.

Examples of background shaders are plain and graduated colors, complex patterns, and images generated by previous renderings or scanned from a photograph. When an image is used as a background, simple compositing functions may be performed so that, for example, a rendered image of a building may be superimposed upon a photograph of the site in order to create a more representative visualization. Backgrounds provide a means of improving the visual complexity of an image at a low computational cost.

When the color of the visible surface at a pixel has been evaluated, the current fore-ground shader may be invoked to perform image ‘post-processing’ operations to modify the color in someway before it is output as the pixel value. Examples of such operations are attenuation to simulate atmospheric phenomena such as fog, or depth cueing.

Only a single background and a single foreground may be active at any time, and when a background or foreground is activated, it becomes the ‘current’ one, replacing any other of that class that may have been previously active.

Interesting effects can be achieved simply by mixing two other backgrounds thanks to the "mixed" background shader, to which two other background shaders are specified, along with a mixing ratio.

***Note**: Background and Foreground shaders are not assigned to objects but to the entire scene. No objects must therefore be selected when you activate them.*

Background shaders

Let's now examine in detail shaders of the Background class.

For each shader we provide synopsis, arguments, related default values, and description.

You can become more familiar with each shader by modifying parameters.

"clouds"

Class **Background**

Synopsis A cloudy appearance.

Description A background giving a cloudy appearance.

The color of the clouds and the background (sky) may be specified in arguments "**clouds color**" and "**background color**" respectively.

The detail or complexity of the texture may be controlled by means of the "**detail**" argument. A value of 1 will give a simple outline of the clouds, while a larger value such as 5 or 6 will give fine detail on the cloud outlines.

An overall scaling factor is provided in argument "**scale**"; increasing this value will make the clouds appear larger.

"graduated"

Class **Background**

Synopsis A graduation between two colors from top to bottom of the image.

Description A background providing a graduation from the top of the image to the bottom of the two colors specified as arguments "**top color**" and "**bottom color**" respectively.

"image"

Class **Background**

Synopsis A background of an image read from a file.

Description A background shader that maps an image.

The name of the file containing the image data is provided as a string to parameter "**file name**", which can be in any format for which the appropriate image driver is installed.

If the supplied image is larger than the image that is rendered, then its top left corner will appear as the background. If the supplied image is smaller than the rendered image, then it will repeat itself both horizontally and vertically to fill the image.

"mixed"

Class **Background**

Synopsis Allows the effect of two other background shaders to be mixed.

Description A background shader that mixes the results of two other background shaders ("**base shader**" and "**mixed shader**") according to the ratio given as parameter "**mixing ratio**".

Either of the two input shaders may themselves be mixed shaders, giving the possibility of infinitely complicated mixtures of shaders.

Note: *At least one of the specified shaders must have a non-zero contribution, or black will result.*

"plain"

Class **Background**

Synopsis A plain colored, uniform background.

Description A plain background with a single, uniform color. The color is supplied in argument "**color**".

"ray cube"

Class **Background**

Synopsis Background shader which produces correct results with reflections and refractions.

Description A problem with ordinary background shaders is that they do not handle reflections or refractions. This is because all pixels at which the background is deemed to be visible are shaded the same way, using a background that is located only in image space, rather than in real 3D space. Thus, for example, a mirror, which should reflect the background *behind* the viewer, in fact shows the background exactly as it would be if the mirror were not there (i.e. all the mirrors become transparent). This would even be the case if the mirror had geometry behind it, so the effect becomes one of a hole in the geometry. This shader allows two other background shaders to be used, one for the areas of background directly visible to the viewer, and another for reflections.

Thus the "**primary shader**" is used for primary rays that do not intersect any geometry, while the "**secondary shader**" for reflection rays which do not intersect any further geometry (i.e., reach the background).

If neither of the shaders is defined, then black is used as the default background color.

This shader is very useful for rendering images where the directly visible background is supposed to be replaced in post-production. The environment background would be used for reflections, and the plain background for the directly visible background, which can then be replaced by an image. This may be useful when producing animations.

Note: ***solidThinking*** *does not distinguish between those secondary rays that are caused by reflection and those caused by refraction. Thus refracted rays that do not intersect geometry will be rendered according to the "secondary shader" background shader, which may produce odd results in some cases.*

"scaled image"

Class **Background**

Synopsis A background of an image read from a file, and scaled to fit into the viewport. The image may also be rotated.

Description The name of the file containing the image data is provided as a string to parameter "**file name**", and it can be in any format for which the appropriate image driver is installed.

The image is automatically scaled to fit the viewport. If the parameter "**keep aspect**" is set to TRUE, then the pixel aspect ratio of the image is preserved (which may result in a portion of the image being cropped).

The image may also be rotated through either +/-90, +/-180, or +/-270 degrees via the parameter "**rotation**". Positive numbers indicate a clockwise rotation, negative an anti-clockwise.

“two planes”

Class **Background**

Synopsis Background shader that produces correct results with reflections and refractions.

Description A problem with ordinary background shaders is that they don’t handle reflections or refractions. This is because all pixels at which the background is deemed to be visible are shaded in the same way, using a background which is located only in image space, rather than in real 3D space. Thus, for example, a mirror, which should reflect the background behind the viewer, in fact shows the background exactly as it would be if the mirror were not there (i.e. all the mirrors become transparent). This would even be the case if the mirror had geometry behind it, so the effect becomes one of a hole in the geometry.

This shader allows two background images to be located in world space (one in front of the observer and one behind). These images can then be ‘reflected’ in any mirrors, thus producing the correct effect (refraction is also correctly handled). This is done using ray tracing.

The two background images are generated from two ordinary background shaders, which may or may not be the same. The two background images are located on infinite planes positioned at "**distance**" front and back of the eye point. If the distance specified is nearer than the far clipping plane, then the distance to the far clipping plane is used (hence the default value of 0:0).

The images are parallel projected onto these planes. In the case of procedural shaders such as "clouds", any point on the infinite plane should give rise to a sensible value, but in the case of images (or the "graduated" procedural shader), a method is needed to decide which are the color areas on the plane that lie outside the background normal area.

According to the style of shader, this could be done by ‘tiling’ the image repeatedly, or by ‘smearing’ it, which simply means that the color outside the scope of the original background image is the same as the point on the edge of the image. The extrapolation method may be set differently for each of the two shaders using the parameters "**front extrapolation**” and "**back extrapolation**". These should take one of the following values: "none", "smear" or "tile". For example, for "clouds" the correct extrapolation method would be "none". For "graduated" it would be "smear", and for an image background it would be "tile". Obviously using an image file which is tile-able will give the best results.

Foreground shaders

Let's now look at detail shaders of the Foreground class.

For each shader, we provide synopsis, arguments, related default values, and description.

You can familiarize with each shader by modifying its parameters.

"depth cue"

Class **Foreground**

Synopsis A linear attenuation within a specified depth range.

Description A foreground shader that adds a background color specified by parameter "**background color**" according to the distance of the surface from the viewer.

At distances less than that given as "**near**", no background color is added. At distances greater than "**far**", the full background color is returned. Between the two distances, the output is linearly interpolated between the surface and background colors.

The effect is that depth information in the geometry is enhanced in the final image.

"fog"

Class **Foreground**

Synopsis An exponential attenuation for atmospheric simulation.

Description A foreground shader that implements atmospheric absorption to create a 'fog' effect.

The fog color is given by "**fog color**" argument.

Increasing "**distance**" value moves the fog farther away from the viewer.

"foglight"

Class **Foreground**

Synopsis Volumetric lighting with point or spot light sources.

Description Atmospheric scattering from point and spot light sources.

A constant distribution of particles in the atmosphere is assumed.

Volumetric shadows are not implemented by this shader.

"ground fog"

Class **Foreground**

Synopsis Simulates a layer of fog whose density decreases with altitude.

Description This shader is used to create more subtle fog-like effects than the standard "fog" shader. With this shader the fog density decreases exponentially along a user specified axis.

The parameter "**fog height**" sets the decreasing rate (if this is set to zero or a negative value, the effect is the same as with the standard "fog" shader).

The parameters "**ground point**" and "**ground normal**" define the place where the fog lies. It is assumed to be the plane z=0 by default. Note that fog does not exist at all below the ground plane, so in the default case there is only fog in the z>0 half space.

"light scattering"

Class **Foreground**

Synopsis Atmospheric scattering of light, with attenuation. A more general version of the simpler "fog light" shader.

Description This foreground shader implements atmospheric scattering and attenuation. It uses true volume sampling in order to generate accurate and consistent results, including volumetric shadows.

The camera and the geometry can move freely without restrictions through the atmosphere.

Note that the shader works with *point*, *projector* and *spot* lights, and you must enable their scattering option in order to obtain results.

The shader produces volumetric shadows (both soft and ray traced), has 5 anisotropic scattering models, allows the fractal noise used to be altered, and has a straightforward speed versus quality control (the sampling rate).

There are quite a few control parameters in order to provide a high degree of flexibility, and these are listed below:

"fog density" Overall particle density of the fog. Increasing this value makes the atmosphere more visible because a dense medium scatters more light than a clear one. If no attenuation is used, it has the same effect as scaling up the light source intensity. Default value is 1.0. The fog density needed in a specific scene will depend on the scene size and intensity of the light sources.

A scene with objects thousands of units big illuminated with powerful spotlights and having an ISL fall-off will need a pretty high-density setting such as 10000. A scene with object sizes near a unit and light sources with a unit intensity will do ok with a density near 1.0. If everything looks too saturated because of the high density, some attenuation could possibly help.

"samples" Number of samples per ray. A higher number of samples gives more accurate results and it is the safest way to alleviate volume aliasing artifacts, which are especially annoying in the shadows cast by objects in the atmosphere. Unfortunately, high sampling rates are computationally expensive. If there is not a lot of shadow complexity in the scene, a low number of samples such as 5 is enough as a fast preview, and some high contrast projector lights might need as many as 50 samples. Default value is 20. Note that this is the initial sampling rate. Due to adaptive antialiasing (if " AA level" is greater than zero), we could probably end up taking lots more samples.

"max depth" Maximum distance along the ray. The ray is clipped with a sphere centered at the ray origin and a radius equal to max depth, so any part of the ray at a further distance is ignored in the computations.

This control is provided to the user to help in the speed/aliasing problem. The higher the value, the more likely it is that we get aliasing problems with rays getting lost in the sky, because the limited number of samples used has to cover a very long distance and a lot of detail could be missed between samples. A very low value near 0.0 would result in an image lacking any volumetric effects. Default value is 1000. This parameter strongly depends on scene size.

"falloff threshold" The shader will clip the area of influence of the light sources with a sphere centered in the light source location whose radius depends on this parameter, so no volumetric rendering will be performed outside the sphere of influence. The idea is that, due to inverse-square fall-off, the intensity of light coming from a given light source will be very small beyond some distance, and it would be unnecessary to take a lot of samples in that low contribution area. Default value is 0.001.

This value is strongly dependent on scene size and light source intensity. If the threshold is too high, the spotlight will appear clipped and a sharp discontinuity in the beam will be visible. You should use the biggest threshold you can go along with (i.e., the discontinuity should not be too noticeable), so the sphere of influence will be the smallest possible.

"AA level" Depth of recursion of the volumetric adaptive antialiasing algorithm. If two volume samples differ in their shadow information (shadowed/not shadowed) or the scattered light at the two sample points differs in more than some specific amount ("AA threshold"), the shader will perform binary recursive adaptive supersampling, placing more samples in segments of the ray where interesting things are happening; this requires a much smaller number of samples to eliminate aliasing than the brute force approach of a huge initial sampling rate.

With a recursion level of 0 no supersampling will occur, and we can be sure that no more than the specified number of samples will be taken. With a recursion level of 1, the shader will use 2× samples as an absolute maximum, and so on. You will rarely need a recursion level greater than 3 or 4. Default value is 2.

"source attenuation" This parameter affects the incoming light from the light sources (typically a spotlight) at a volume sampling point. It makes the light fall-off much smoother. This has a subtle effect in the spotlight's beam, helping the beam die much more quickly, and making the clipping introduced by the "fall-off threshold" parameter much less noticeable.

The exact amount of attenuation to use in different scenes depends mainly on scene size. Users will have to experiment with the settings until the desired effect is achieved. Use these three new attenuation controls subtly, starting with low values such as 0.01 and slowly turning them up until typically 0.1 or 0.5. If you use a very high source attenuation value, it's possible that you won't get any scattered light at all; the spotlight beam will be very dim. Default value is 0.0 (no attenuation).

"surface attenuation" This parameter affects the underlying surface color under the pixel. The light coming from a solid object visible in the pixel is attenuated by the fog between the intersection point and the eye. Objects farther away from the eye will appear darker. This computation is performed once per pixel, outside of the volume sampling loop.

Use these three new attenuation controls subtly, starting with low values such as 0.01 and slowly turning them up until typically 0.1 or 0.5.

If you use a very high surface attenuation value, all the geometry under the pixel will be black, since the fog will strongly absorb almost all the light reaching the eye from the object surfaces. Default value is 0.0 (no attenuation).

"volume attenuation" Scattered light coming from each volume sample along the viewing ray is also affected by absorption since it has to travel through the atmosphere before reaching the eye. This has the effect of making the beams of light darker the farther away they are from the eye, being the volumetric counterpart of the previous parameter, "surface attenuation".

Use these three new attenuation controls subtly, starting with low values such as 0.01 and slowly turning them up until typically 0.1 or 0.5. If you use a very high volume attenuation value, your beams of light will only be visible when located very close to the camera, and they will get progressively dimmer and dimmer the farther away from the camera. Default value is 0.0 (no attenuation).

"scattering model" Selects a scattering model out of five possible ones: "isotropic", "Rayleigh", "hazy Mie", "murky Mie" and "Henyey-Greenstein" scattering.

Strong anisotropic forward scattering such as the murky Mie model manifests itself when looking directly at the light sources. A very small amount of light is scattered in directions perpendicular to or backfacing the incoming light vector. All models are normalized to lie inside the unit sphere when represented in polar coordinates. Default model is isotropic (view-independent scattering). Note that the default is used if an unrecognized string is given.

"eccentricity" This is the eccentricity parameter of the ellipse used in the Henyey-Greenstein model. It has no use at all with the other scattering models. The parameter has to be in the [-1, 1] range. A value of 0 produces isotropic scattering, while a high positive value produces strong forward scattering, and a high negative value produces strong backward scattering. Default value is 0.0.

Note that values very close to +/-1 will give very narrow anisotropic distributions in the forward/backward direction, so you will have almost no scattered light at all. Sensible values lie around 0.5 for anisotropic forward scattering (negative values give backward scattering).

"noise amplitude" Fractal noise perturbation amplitude. The fog density is modulated with this fractal noise signal. When it is set to zero, no fractal noise computations are performed at all (faster). Default value is 0.0.

"noise scale" Relative size or scale of the noise. Depends on scene size. Default value is 1.0.

"noise octaves" Number of octaves (also known as frequencies, scales) in the fractal noise signal. The more octaves you have, the higher the detail in the noise, and the longer it will take to render.

Remember that the computation time is linear in the number of octaves. Default value is 2. You will rarely need more than 4 octaves.

"noise gain" Noise contrast control, in the range [0, 1]. High values result in sharp and fast transitions, while low values smooth out the noise signal. A value of 0.0 turns the noise perturbation off. Default value is 0.5 (the gain filter is by-passed).

Most of the times you will not need to touch this control and the default value should always look fine. It can make the turbulence in the fog much more subtle or stronger. A value of 1.0 would provide a very high contrast noise signal, almost a binary step function switching from 0.0 to 1.0. Sensible values lie in the range [0.2, 0.8].

"snow"

Class **Foreground**

Synopsis Gives the effect of snow flakes falling in front of camera.

Description A foreground shader that gives the effect of snow flakes falling in front of the camera.

The effect is produced by two overlaid planes of randomly dispersed 'snowflakes'. The scale of each of the two planes is specified by the parameters "**near scale**" and "**far scale**".

The size and density of the flakes is given by the parameters "**flake size**" and "**flake density**", and the irregular appearance of the edges of the flakes is controlled by the "**noise amplitude**" and **"noise scale**" parameters (these last two values should lie in the range 0.0 to 1.0).

Finally, the color of snowflakes is specified by the parameter "**flake color**".

5.7 Sketch Styles

Sketch rendering is used to create stylised presentations. Styles available include cartoon, pencil, ink and various paint effects.

In contrast to photorealistic rendering, the aim of Sketch rendering is not to use the computer to produce an image which looks as close as possible to a photograph, but, instead, to use the computer to produce an image which is more like a pencil sketch, ink drawing, or
painting.

Sketch rendering allows you to communicate different aspects of your designs, to different people, at different stages of the design process.

For example:
• at very early stages of the design process, photoreal images are often unlikely to look convincing—however images produced using Sketch Rendering can be compelling;
• the shape of a design can be effectively communicated without the viewer's eye being drawn to issues such as the materials used;
• designs may be presented in a way which does not give the impression of finality which photorealism sometimes gives;
• interesting marketing imagery can be produced. Sometimes they can be more eye catching than the photoreal images.

In order to use Sketch Styles you must first choose **Render ▸ Options** and select *Sketch rendering* as **Method**.

Then, open the **Shading** panel (**Managers ▸ Shading)** and select the **Image** tab. Right click **Sketch Styles** to choose the effect to apply.

The following pages outline each sketch style available.

"cartoon"

A classic cartoon style effect, where silhouette and boundary edges are picked out with bold lines, and colors are somewhat simplified and stylised.

It is possible to control the color and width of the lines via shader parameters. The colors which are painted between the lines will depend on the materials, but are quite flat (i.e., very little tonal change as you look across a surface) to reflect the simplified palette employed by cartoon artists.

This style uses hidden-line.

"color wash"

As in the "cartoon" style, edges are picked out with lines, and the spaces in between are filled with simplified color. In the case of "color wash", however, the colors are less cartoon-like, and can be washed-out to look much more pastel and less intense.

The width and color of the lines is still specified via shader arguments, but it is also possible to control the degree of color variation across objects, and the degree to which the colors are washed.

This style uses hidden-line.

"contour"

Creates a very ‘painterly’ effect, with strokes and swirls of color which reflect and represent the orientation of the underlying geometry. The effect may also be interpreted as being akin to using colored pencils to shade the image (with the direction of the pencil strokes matching the underlying geometry at all times).

Normally the colors of the strokes would be taken from the color of the objects being represented—however this can be over-ridden so that a particular ‘line color’ is used.

In this case the effect will be similar to a pencil drawing done using a single color of pencil.

It is also possible to alter the length of each of the strokes, and density, or weight, of each stroke. A low value for the line or stroke length will lend a finer, more detailed appearance to the final image; a low value for the density of the strokes or lines will result in more of the background, or paper, color showing through.

The ‘paper’ or ‘canvas’ onto which the lines are drawn can be defined via a background shader. Therefore it is important to choose a color for the lines which will show up against your background!

This style does not use hidden-line.

"hand drawn"

A style in which objects are rendered using lines, but where the lines appear to be made up individual strokes.

The color and width of the lines can be controlled by the user, as can the degree from which each stroke varies from true, the smoothness of the strokes, and also the degree of taper from the start to the end of the stroke. All of these factors can be controlled via shader arguments, allowing the effect to be suited to different models at different resolutions, with different drawing styles or pencils or pens.

The ‘paper’ or ‘canvas’ onto which the lines are drawn can be defined via a background shader. Therefore it is important to choose a color for the lines which will show up against your background!

This style uses hidden-line.

"hatch"

This style produces an effect similar to a shaded pencil drawing, in either monochrome or color. The image is made up of a series of pencil ‘strokes’ all in a similar orientation (short strokes, from bottom left to top right in direction, as though a right handed artist were shading roughly with a pencil).

All of the strokes are to provide shading—no ‘lines’ are drawn. The user can control the color, width and length of the pencil strokes, as well as how straight or curved the strokes are, and their density. A low value for the density of strokes will result in a very sparse looking image, with plenty of ‘paper’ showing through, whilst a higher density will result in dark regions being almost completely shaded (note however that lighter regions will still show up as the ‘paper color’, or background).

It is also possible to specify that the color of the underlying geometry (derived from material information) be used for the color of each stroke, creating a ‘colored pencil’ or perhaps ‘pastel’ drawing effect.

Note that as the current background shader will show through where there are gaps between each of the shading strokes, so if the density of strokes is not high it will be important to choose a sensible background color or image for your ‘paper’.

This style does not use hidden-line.

"ink print"

This style can be thought of as almost a photographic negative of other line-based styles.

The entire image is rendered in a given ‘ink color’, save for the lines, which are simply left as ‘gaps’ in the ink color.

The ‘paper’ or ‘canvas’ onto which the ink print is placed can be defined via a background shader. Therefore it is important to choose a color for the ink which will show up against your background!

This style allows the user to specify the ink color and the size of the gaps, as always via shader arguments.

Note that this style uses hidden-line.

"lines and color fill"

This style combines outlines with shading strokes, allowing control over the full range of effects which sketch rendering is capable of.

We recommend that this style is used by expert users only, perhaps to create instances of styles which would otherwise be difficult or impossible to achieve.

"lines and shadow"

This style combines a simple line representation of geometric objects with a (monochrome) shaded effect for areas in shadow. The result is very much like a pencil drawing where edges and shape are captured first using lines and then some shading work is done with the edge of the pencil.

The user can control the color and width of the lines, as well as the color used for shading in the shadow regions (via shader arguments).

The ‘paper’ or ‘canvas’ onto which lines and shadow are drawn can be defined via a background shader.

This style uses hidden-line.

“mosaic”

This has the effect of rendering the image as though it were constructed as a mosaic of small colored tiles. The size and shape of the tiles can be controlled via shader parameters, as can the size and color of the gaps between the tiles. It is also possible to use the background color as the color for the gaps between the tiles, or to blend a background and a given gap color.

If desired, it is possible to leave areas of exposed background in the scene untiled; in other words only to produce a mosaic pattern for the geometry, and apply some other pattern or image entirely for the background.

This style does not use hidden-line.

"oil painting"

Creates an effect reminiscent of some impressionist painters, with individual blobs of paint (which could be brush or palette-knife strokes) of a single color being applied across the canvas, so that the overall image only resolves itself if you view from far enough away.

The brush strokes are always slightly circular, roughly ”C” shaped marks. The shape of the strokes is not affected by the orientation, proximity, or color of the underlying geometry which the stroke is representing.

The effect looks quite good with image scenes that have a strong shape and, do not rely too much on detail to convey their meaning. Scale, and how the detail relates to scale, is clearly quite important, as if the size of the brush strokes means that important detail is lost then the results may be unimpressive.

It is possible to alter the size of the individual marks, and the number of them in the whole image (it is important to choose values for these parameters such that any detail required in the final image will be resolved, without losing the ‘painterly’ effect).

It is also possible to control the degree to which the marks obscure the background color: it is possible to fill the canvas entirely, with marks which overlap each other, or to leave gaps between the marks, creating a ‘lighter’, less over-done feel.

This style does not use hidden-line.

"rough pencil"

Produces an effect which as though an artist had gone over each line several times, each with a small margin of error. The style is well suited to situations where a design is still quite tentative, as the effect is of a very rough, quick, pencil sketch.

The color, width, and number of lines can be controlled via shader parameters, as can the ‘wobbliness’ of lines (how far each line can deviate from true) and the ‘curliness’ of lines (how far from straight or jagged each line is).

Tthe ‘paper’ or ‘canvas’ onto which the lines are drawn can be defined via a background shader. Therefore it is important to choose a color for the lines which will show up against your background!

Note that the current background shader is interpreted as being the ‘paper’ or ‘canvas’ onto which the lines are drawn.

This style uses hidden-line.

"soft pencil"

"soft pencil" mimics the effect of a pencil drawing done quite softly, with quite a soft pencil, so that in places the pencil leaves hardly any mark.

The color and width of the pencil line can be controlled, as can the sharpness of the lines. It is also possible to control the extent to which the pencil line appears to fade in and out as the pencil travelled over the paper.

Note that the current background shader is interpreted as being the ‘paper’ or ‘canvas’ onto which the lines are drawn.

This style uses hidden-line.

"stipple"

This effect causes the image to be rendered as a series of irregular dots, or stipples.

The effect is similar to how photographs are reproduced in newsprint, only more pronounced.

By default the stipples are all rendered in one color (black by default, but the ‘ink’ color can be altered). It is also possible to use the color of the underlying geometry for each stipple - if this option is chosen, the effect becomes reminiscent of the ‘pointillism’ school of impressionistic painting, where scenes were rendered by points of color or light.

This style does not use hidden-line.

5.8 Radiosity

The Radiosity technique involves simulating the lighting distribution throughout an environment, using a physically accurate lighting model.

Radiosity is in no way responsible for the production or rendering of an image. It takes geometry as its input, and generates geometry as its output - the output geometry is an alternative representation of that supplied as input, but subdivided into small 'elements', and with radiosity values attached to each vertex. This geometry can be used for rendering employing any of the standard rendering methods available in ***solidThinking***.

The idea underlying the radiosity technique is to model the distribution of all the light energy in an environment, by applying the conservation of energy at every surface. It should be understood that the radiosity algorithm solves the lighting problem, but not the visibility problem. Once the light distribution has been generated, images can be rendered using standard visibility and shading algorithms.

The advantage of radiosity as a technique is that it accurately models the type of environments where there are matte surfaces which diffusely reflect light in all directions, and onto other matte surfaces; the inside of a building is a very common example of such an environment. It should be noted that many surfaces in an interior environment are lit by no direct illumination at all, and are visible only by light reflected diffusely from other surfaces.

Also, a surface illuminated indirectly may appear to be of a different color than it would appear if lit directly, since color from one surface can “spill” or “bleed” onto another, particularly if bright colors are placed next to more subdued hues.

A number of important effects are simulated by radiosity techniques. Radiosity is able to account for the area of light sources, resulting in accurate soft shadowing. Also, by accounting for secondary illumination, radiosity correctly computes “color bleeding” effects and the indirect illumination of parts of an environment that are not directly lit by a primary light source.

Radiosity is therefore an important technique for architects, builders and lighting engineers, as well as general computer graphics users, and anyone who is interested in creating realistic images.

The disadvantage of the technique is that it is costly in terms of computer time and memory. It is however the only technique that can reproduce physically accurate lighting conditions in a large class of common scenes.

How the Radiosity Technique Works

In the past, other techniques have sought to approximate the effect of diffusely reflected light in a scene by applying a constant 'ambient' value of illumination indiscriminately through the entire scene. Radiosity is essentially a technique which correctly computes the distribution of this 'ambient' light, accounting for light reaching a surface after being reflected from one or more diffuse surfaces.

Radiosity techniques originated in thermal engineering, where they were used to simulate radiative heat transfer between surface elements. In recent years, computer graphics researchers have shown how radiosity can be used to simulate the transport of visible radiation in an environment. The physical accuracy of radiosity solutions results in images with a stunning level of realism. If physically correct input data is provided, then the results are faithful representations of the real world. This allows aesthetic judgements to be made from radiosity images, and also allows quantitative measurements of illumination levels to be obtained from the simulation results.

The technical definition of the term 'radiosity' is that it is a measure of radiant energy, in fact "the amount of energy leaving a surface, per unit area, per unit time". The principle underlying the technique is to divide all the surfaces in the model into small areas (called 'patches'), and to calculate this 'radiosity value' for each. This gives a value for the 'brightness' of each patch, which can then be used to produce an image made up of all the patches visible from the viewpoint, each rendered lighter if they are 'bright' or darker if they are 'dim'.

Computing Radiosity Values

To enable radiosity computation, you must first open the **Shading** panel by using either the **Managers ▸ Shading** command.

Once opened, the **Shading** panel displays six tabs (Surface, Light, Image, Rendering, Output and Globals) on the upper left corner. Select the **Rendering** tab: the first column on the left displays the Rendering chain, which lists, among others, the Radiosity option and gives you access to the controls that are displayed in the column beside.

Performance is optimized wherever possible, in accordance with the settings specified by the user, so that the required results are achieved as quickly as possible. It is only when very high quality results are required that a great deal of time will be taken.

Units of Length
One important point to note about the radiosity processor is that by default all dimensions are assumed to be presented in *meters*. The radiosity processor needs to be told explicitly what units should be used.

Defining Lights for Radiosity

Light sources are specified in ***solidThinking*** by the use of light source shaders. Radiosity divides the light sources in an environment into two groups:

- primary sources (defined by light source shaders);
- secondary sources (defined by materials).

Initially, only the light sources have positive power; all other surfaces have zero radiosity. The light sources provide the initial energy input for the environment and are known as the primary sources.

Secondary sources are surfaces that have been illuminated by primary sources and/or by other surfaces. These, due to their unshot radiosity, can now act as additional sources in the environment; thus they 'reflect' light, by re-radiating a portion of the energy incident on them. The behavior of secondary sources depends on the material they are associated with.

Once the energy from the primary sources has been shot, all surfaces are treated as both emitters and receivers/reflectors of light energy.

In ***solidThinking*** primary light sources (i.e., light shader types) supporting radiosity are:

"area" Can be used to represent many different light sources in a model; examples include strip lights, neon signs, shaded lamps, or windows with blinds drawn. Groups of area light sources can be used to create a complex single source.

"point" Used to represent sources that radiate light equally in all directions, such as a simple light bulb.

"spot" Used to represent sources that radiate light in a circular cone, directed along a particular direction, usually with light intensity falling off as we move away from the cone's axis of symmetry.

"goniometric" Used to represent sources whose intensity distribution (i.e., how much light is emitted in any one direction) cannot be easily defined via some formula, as is the case with spot and point lights. Goniometric lights rely on being handed a file, which details how much and in which directions light is emitted. The lighting industry produces exactly this sort of data for the luminaires that they manufacture.

"sky" Used to represent the contribution of indirect sunlight, i.e., light from the sky itself. Essential for daylight simulation.

"distant" A distant light can be used to represent a light that is located far away, such as the sun (all rays of light from such a source will be parallel).

The Radiosity module does not support ambient light source, which is therefore ignored during the radiosity simulation.

Rather than using the standard "ambient" light source shader, ambient lighting inside the Radiosity module should be controlled by enabling the **Ambient** option, which evenly distributes all the not-yet-accounted-for energy in the scene over all surfaces.

This provides a much better approximation of ambient lighting conditions after only a few iterations of the radiosity solution.

Of course, all ‘ambient’ lighting effects are only approximations to fully realistic indirect lighting. This can be modeled more accurately by running the radiosity simulation over a large number of iterations, so that all the important secondary sources are accounted for.

"area light"

An area light source emits light in a hemisphere of directions. The radiance leaving any point of the source, in any direction within the hemisphere, is constant. Area light sources work only when Radiosity computation is enabled.

Name "area"

Class **Light Source**

Synopsis Light energy from an area source.

Description A light source shader simulating a glowing object in the scene. Each area light should be associated with an object in the scene, and therefore no positional or other geometric information need be specified explicitly for the source. In fact, as with any other surface, area light sources need to have a material assigned to them, as not only do they emit light themselves, but they may well have to reflect light, from other sources, which is incident upon them. The radiosity processor always automatically recognizes the defined area sources.

The "**unshot**" argument represents the unshot radiosity value, in other words, the amount of light energy, per unit area, which the area source will emit. This is specified per unit area of the source for each of the red, green and blue components. The SI units for radiosity are Watts per square meter.

Because the radiosity of an area light source is defined in terms of its area, relatively small alterations to the geometry to which it is attached could lead to large changes in the lighting conditions. It is worth calculating the area of a light source before deciding what radiosity value should be assigned to it.

An alternate way to specify power and color of an area source is defining power units ("undefined", "watts", and "lumens") and setting directly power ("**power**" parameter) in these units. In this case, as we specify the total power, the impact of changing the source geometry will be not so critical.

The source color, when using defined power units, is specified by "**color temperature**" and can be additionally filtered by given "**color**" parameter. If "color temperature" is not set (equal 0:0), then the "color" is used for color definition.

Area light sources are associated with polygonal geometry, which by default is single-sided. This means that an area light source only radiates energy in the direction away from the 'front' of the surface.

The parameters "min lod", "max lod" and "noise factor" are effective only during direct rendering (ignored by radiosity). Meanings are:

"**min lod**" - minimal level of detail of area source decomposition. The usual range is [0:0; 1:0] although values greater than 1:0 are allowed. The value 0:0 means minimal level of decomposition. This parameter determines the initial sampling for lighting calculation and visibility analysis. If it is too low, then shadow boundaries will not be reproduced correctly. If it is too high, then rendering times will be excessive. Practical values are between 0:2 and 0:6.

"**mid lod**" - this parameter is similar to "min lod" but it does not cause actual decomposition of the source for lighting calculations; it determines instead minimal level of decomposition for visibility analysis. If should be no smaller than "max lod" and, if it is actually smaller, then is defaulted to "max lod". Controlling the level of decomposition with this parameter makes sense if we use shadow maps, or shading calculations are extremely costly and we want to limit number of actual sample but at the same time we want to perform sufficient shadow analysis.

"**max lod**" - maximal level of adaptive decomposition of source when rendering areas of penumbra. If set to the same value as "min lod", no adaptive decomposition is performed and penumbra areas may exhibit aliasing artifacts. The range is [0:0; 1:0] although values greater than 1:0 are allowed. The practical range is [0:5; 1:0].

"**noise factor**" - specifies level of random disturbances for visibility samples. Value of 0:0 means no noise. Value of 1:0 means maximal noise. The range is strictly [0:0; 1:0]. The noise may be used as cheaper alternative for antialiasing penumbra areas than adaptive refinement (controlled by max lod parameter).

"**shadows**" - switches shadows on or off for this source.

"**shadow resolution**" - resolution of shadow map associated with this source. If set to 0, then ray traced shadows are used. See below.

"**shadow quality**" - quality of shadow map sampling expressed as square of the number of point samples to be taken to approximate shadow for given shaded point.

Defining Materials

As already noted, radiosity technique models diffuse reflections between different surfaces in the scene. In order to do this, the fraction of the incident light that each surface reradiates needs to be defined. This is achieved by making use of material shaders. The intention is that materials shaders can be set up in such a way that surfaces are fully specified for both the radiosity simulation and for later rendering, without the need for alteration.

Two information are required for the calculations of radiosity simulation. Firstly, the color of each surface needs to be known, so that effects such as color bleeding are modeled; Secondly, the overall fraction of the light incident on each surface, which will be diffusely re-radiated by that surface, must be specified.

Only two classes of materials can be used during the radiosity solution: color and reflectance. The light re-radiated by a surface is calculated by the multiplication of the diffuse reflection factor provided by a correctly set up reflectance shader and the color value provided by the color shader. Any of the other classes may also be made use of: they will be ignored by Radiosity, but used by those rendering methods to which they apply, when the radiosity solution is rendered.

Once the radiosity simulation is complete, the result can be displayed by rendering the processed geometry as required. The solution can be rendered in any supported rendering methods, such as preview, full, and so on. To enable realistic shading effects to be obtained for these modes, it will be necessary to use materials that will create the relevant effect. Materials shaders such as procedural texture shaders, texture maps, bumpmaps, mirrors, and so forth, can be used to give added realism to the finished image if it is rendered using a high-quality rendering method.

Rendering using a photo-realistic method will allow the above effects, as well as the radiosity solution, to be incorporated into the final image, allowing very high quality images. These will not only exhibit realistic lighting conditions, but also reflections, textures and surface imperfections which all contribute to making an image look real rather than artificial.

5.9 Final Gather

The new release introduces Final Gather: an exclusive technique that can be used with or without Radiosity and which will calculate the effects of diffuse secondary illumination (i.e. bounced light) in a scene.

It involves computing the incoming illumination at various points in a scene and storing this data in a cache. Information stored in the cache is then used to determine how illumination from other parts of the scene contribute to the illumination at the point being shaded.

Thanks to Final Gather extremely accurate and well-lit images can be produced very quickly.

Final Gather can be used to construct an Irradiance Cache which, in turn, can be used for a variety of purposes, including:
• Simulating ‘one bounce’ global illumination.
• Hiding mesh artifacts in a coarse radiosity solution, whilst at the same time illuminating the entire image by the unshot energy in the radiosity simulation.
• Accelerating ‘dome’ light source shaders, such as "sky" and "environment".

What is irradiance? *Irradiance is a color; it measures light energy incident upon a surface from anywhere in the hemisphere above the point of interest. Irradiance is the incident light energy per unit area, per unit time. Typical units are Watts per square metre. An* ***Irradiance Cache*** *is a data structure which stores irradiance values allowing us to re-use its contents. If we know the irradiance in the middle of a field is X, then we can be pretty sure that when the next pixel wants to know the irradiance a few centimetres away, the value will be something very close to X. We don’t need to perform another*

Final Gather operation; we can simply grab a few nearby values from the cache and use interpolation to get our result.

To enable the Final Gather method, you must first open the **Shading** panel by using the **Managers** > **Shading** command.

Once opened, the **Shading** panel displays six tabs (**Surface**, **Light**, **Image**, **Rendering**, **Output**, and **Globals**) on the upper left corner. Select the **Rendering** tab: the first column on the left displays the **Rendering** chain, which lists, among others, the **Final Gather** option. Checking the radio button on enables the Final Gather computation and allows you to modify parameters that are displayed in the column beside.

The "**saturate output color**" parameter is used to control how much the Final Gather color is saturated. A value of 0.0 (the default) corresponds to no saturation taking place. A value of −1.0 turns the color pure grey, whilst a value of 1.0 results in extremely rich hues.

"**filter output color**" filters the Final Gather color by the given color.

"**scale output intensity**" provides an easy way to modify the amount of indirect light in a scene by controlling the contribution of the Final Gather to the final rendering.

The "**domelight components**" argument allows you to specify how the contribution of a 'dome' light (i.e., a "sky", "simple sky", "environment" or "simple environment" light) should be modelled, during any final gather processing. The Final Gather method is able to simulate two different types of lighting contribution:
• **Direct** diffuse lighting contribution from light sources that we know have a slowly-varying distribution, due to their extent. These lights are the "sky", "simple sky", "environment" and "simple environment" light source shaders, collectively known as the dome lights, since they form a massive dome over the scene being rendered.

• **Indirect** diffuse lighting contribution (e.g., modelling the way uplighters illuminate a floor, as their light bounces off the ceiling).

***Note**: "sky", "simple sky", "environment" and "simple environment" dome lights has a "lighting components" shader argument which allows you to specify how the light's contribution should be modelled, during any final gather processing:*

*• **Gather Direct**. Model the dome light's direct contribution using the final gather module. So, when a final gather sample ray misses all the geometry in the scene, establish how bright (and what color) the dome light is, in that direction, and return that radiance. If, for a particular final gather sample point, many such rays miss all the geometry in the scene, a correspondingly bright 'sky' contribution will result. If most sample rays hit opaque geometry in the scene, then the 'sky' contribution will be small. The fastest renders come with this setting enabled and Gather Indirect disabled.*

*• **Gather Indirect**. Model the dome light's indirect contribution using the final gather module. So, when final gathering on the ceiling of a room, spray out sample rays to find the floor and walls and so on, and establish how the dome light illuminates each of these points using the dome light's own parameters. So the dome source will bounce light of the floor and walls, onto the ceiling, where it is captured by the final gather module. Illuminating any point with a dome light can be expensive, so it is worth noting that a very coarse illumination is probably enough for the final gather to do a good job. Whilst such settings would produce a very poor render, if used in the absence of the irradiance cache, they are usually perfectly adequate for indirect illumination. The highest quality renders come from setting enabled.*

*• **Global**. It allows global control over the "lighting components" of any dome lights that happen to be in use. For example, if there is an archive of dome lights, all of which have "lighting components" set to Global, the user can then explore different settings and light sources, without having to constantly mess with the individual lights.*

"**debug**" allows the locations of irradiance samples to be displayed in the rendered image. Samples created during cache initialisation will be shown as green dots by default, samples created during rendering.

The "**number of rays**" parameter determines how many visibility rays are cast into the hemisphere above the point of interest, when performing a Final Gather operation. Lower values give noisier, faster, less accurate results. Higher values give smoother, slower, more accurate results. In scenes with few occluders and near-uniform illumination, large values are not needed. Where we have many occluders, or complex illumination, we have to cast more rays if we are to get a reasonable irradiance estimate from our Final Gather operation.

Sensible values for "**minimum radius**", "**maximum radius**" and "**maximum ray length**" depend on:
- the size of the scene being rendered;
- whether the whole scene is being rendered (or just one region of a larger scene);
- speed/accuracy trade offs.

The "**interpolation quality**" parameter quantifies what 'sufficiently' means in the statement 'if sufficiently many cache samples, are found sufficiently close to the point of interest, then interpolation will be used rather than Final Gather.'

The "**normal tolerance**" parameter controls 'interpolation around corners'. When rendering scenes are primarily composed of planar surfaces, there is little need to fine tune it. If curved surfaces do appear in a scene, but they are not visually important, then large values should be used, in order to increase performance. The parameter specifies an angle in degrees. The range is [0.01, 90.0]. In general it is necessary to have a "normal tolerance" which is greater than the angle between adjacent facets in order to get a smooth appearance on curved surfaces.

Whilst the Final Gather method is able to simulate two different types of lighting contribution (direct diffuse illumination and indirect diffuse illumination), specular contributions are notable by their absence. The reason for this is the directional nature of specular highlights compared to the scalar nature of irradiance. The problem can be avoided by enabling "**evaluate specular**" which evaluate the specular Final Gather by tracing a number of eye rays through each point being shaded and scattering them about the reflection direction, performing shading on each one. The "**specular number of rays**" parameter controls how many such rays are cast for each shading point. If too few rays are cast, then the specular highlight may appear noisy. If too many rays are cast, then the specular calculations will be slow. For a rough surface, more rays should be cast than for a smooth surface (if the same noise level is being sought).

5.10 Feature-Following Anti-Aliasing

To enable the FFAA post-process shader you must first open the **Shading** panel by using either the **Managers ▸ Shading** command.

Once the **Shading** panel is opened, select the **Rendering** tab: the first column on the left displays the Rendering chain that lists, among others, the FFAA option. Check the radio button on to enable it and be allowed to modify parameters that are displayed in the column beside.

FFAA stands for Feature-Following Anti-Aliasing, a post-process shader, which anti-aliases pixel-connected features in an image by high-resolution sub-sampling, while propagating further anti-aliasing between pixels only where necessary. Speed is maximized automatically by super-sampling only on those areas requiring such anti-aliasing.

Class	**Post Process**	
Synopsis	Atmospheric scattering of light, with attenuation. A more general version of the simpler "fog light" shader.	
Arguments	***Name***	***Default***
	"threshold"	0.01
	"relative sensitivity"	4.0

Description The "ffaa" shader provides faithful rendering of connected features extending over several pixels, super-sampling all such pixels efficiently and avoiding the super-sampling of any pixels outside such features. Samples are taken at the corners of a pixel; if this exhibits a difference exceeding a specified “threshold”, then that pixel is considered to contain a high frequency feature and is super-sampled accordingly. The "ffaa" shader adaptively applies high frequency super-sampling to those pixels containing features requiring anti-aliasing, but will not expend time on those which do not.

The "**relative sensitivity**" parameter determines the threshold applied when deciding whether a pixel contains a feature or not. As each successive sample is taken, the cumulative standard deviation of the samples is calculated in a color space allowing for the relative sensitivity of the red, green and blue cones of the human eye. If this standard deviation a) ever exceeds the threshold value, the pixel is taken to contain features, b) never exceeds the threshold value, the pixel is taken not to contain any feature. The "relative sensitivity" parameter should be set high enough to ensure that all pixels containing features are detected (the higher the parameter, the more pixels are reached).

The "**threshold**" parameter should be low enough to ensure that all pixels that contain features are tagged as seeds but not so low that any pixel not containing any feature is incorrectly tagged as a seed.

5.11 Depth of Field

To enable the Depth of Field post-process shader, open the **Shading** panel by using either the **Managers ▸ Shading** command.

Once the **Shading** panel is opened, select the **Rendering** tab: the first column on the left displays the Rendering chain which lists, among others, the Depth of Field option. Check the radio button on to enable it and be allowed to modify parameters that are displayed in the column beside.

This post processing shader lets you simulate camera depth of field effects.

Class	**Post-process**	
Synopsis	Depth of field effect.	
Arguments	***Name***	***Default value***
	"focal length"	50.0
	"aperture"	5.6
	"focus plane"	9.0
	"units"	1.0
	"near focus"	0.0
	"far focus"	0.0
	"far blur"	1.0
	"near blur"	1.0

Description The depth of field effect is controlled either by parameters describing the camera model (lens focal length, aperture stop) or directly (distance to near and far boundary of field in focus, level of blurring in near and far field).

The effect, if desired, can be controlled separately in near field and far field without regard to physical correctness - this gives the ability to create impossible but desirable effects.

The shader also queries the front and back clipping planes. The back clipping plane is used to establish the distance to background. All objects further away than the back clipping plane are set to the distance of the far clipping plane.

The front clipping plane is used to clip nearer objects. If the front clipping plane is nearer to the observer than the focal length, then it is adjusted to focal length with 10 (e.g., if near plane is set on 0.1 mm. while the focal length is 50 mm., then actual near plane will be set on 55 mm). It is so because an image of points closer to the camera than focal distance does not exist.

5.12 Lens Flares

The Lens Flare post-process shader simulates the lens flare effects often produced when cameras are pointed toward bright objects. Components of the effect include central glow, random and regular light streaks, halo and ghost images of lens iris.

Lens flares is a post processing effect: it means that it is applied once the rendered image has been created. To enable the Lens Flares option, just open the **Shading** panel by choosing the **Managers ▸ Shading** command, select the **Rendering** tab, and check the Lens Flare button on.

In the **Parameters** column, the **lens** pop up displays the nine standard “lens types” provided. They are:

"standard" Standard camera. One circular ghost and random light streaks.

"lens35" Mixed light streaks (random and regular), five ghost images.

"lens50" Random light streaks, accentuated bloom, eight ghost images.

"lens105" Mixed random and regular streaks, fuzzy halo, and twenty rounded polygonal ghosts.

"polygon" As "standard", but with rounded polygonal ghost.

"polygon35"	As "lens35" but with sharp polygonal ghosts.
"polygon50"	As "lens50" but with sharp polygonal ghosts.
"spark"	Mixed random and regular streaks, no halo or ghosts.
"star4"	Slightly accentuated bloom, regular streaks.

5.13 Image Output Formats

solidThinking lets you choose the format in which rendered images should be generated. To select the preferred output graphic format, open the **Shading** panel by choosing **Managers ▸ Shading**, click the **Output** tab, and check one of the following options on:

- **Windows Bitmap** (*.bmp*). The standard Windows bitmap format.
- **TIFF** (*.tif*). Tagged Image File Format. Widely used in desktop publishing.
- **JPEG** (*.jpg*). Joint Photographic Experts Group standard. Sliders lets you adjust **Quality** (default is 85) and **Smoothness** level.
- **Quicktime VR Panoramic Movie**. A series of images from a fixed viewpoint, which allow the viewer to freely rotate around the viewpoint and look in all directions.
- **QuicktimeVR Object Movie**. A series of images of an object taken from different angles, which allow the viewer to rotate the object and view all its sides.
- **High Dynamic Range** (.hdr). High Dynamic Range Images can represent a much higher range of intensity values than traditional image formats, which only represent the range of intensity valueswhich can be displayed on a computer display. They have several advantages:
  – they can be used as backgrounds in brightly lit scenes prior to tone mapping;
  – they provide a way to store images which have been rendered using physically based ealistic lighting values, without loss of information;
  – they can be used as environments (via the "environment" light shader) which illuminate scene (allowing digitally created objects to be rendered with lighting conditions exactly matching some real world scene).
- **OpenEXR** (.exr). A high dynamic-range (HDR) image file format developed by Industrial Light & Magic or use in computer imaging applications.

Quicktime VR movies

solidThinking supports the output of QuickTime VR object and panoramic movies. QuickTime VR is a cross-platform standard, developed by Apple, for presenting:

- panoramic movies, a series of images from a fixed viewpoint;
- object movies, a series of images of an object taken from different angles.

The software to view a QuickTime VR movie once created can be freely obtained from Apple via the internet, if you do not have it already. We recommend using the QuickTime 3.0 viewer. More information can be found at the QuickTime VR web site, the URL for which is *http://quicktimevr.apple.com*

QuickTime VR panoramic movies

Panoramic movies are produced from a series of views of a place taken from the same spot, but at slightly different angles.

These images are then combined into a single image to allow viewing. As you pan left and right, or tilt it up and down, QuickTime VR responds to your movements, displaying the correct section of the entire panorama.

The camera position is taken to indicate the initial viewpoint of the panorama. The panorama is taken to be a camera pan through 360 degrees from the initial view.

The middle **Parameters** column of the shading panel shows the following variables:

"Rendering method" Method used to render the frames of the movie. Wireframe and hidden-line are not currently supported.

"Number of images" Specifies the number of individual images that should be rendered to make up the full panorama. This should not be set too low to avoid distortion in the resulting movie. Also, high values should be avoided as the gain in quality becomes minimal. Normally this control should be left at its default setting of 32. Advanced users may wish to fine tune movie generation by slightly changing this value.

"Movie width" Specifies the width of the window that the movie will be displayed in. Default setting is 440.

"Movie height" Specifies the height of the window that the movie will be displayed in. Default setting is 220.

"Movie info" It defines the movie information. Typically used for general information about the movie.

"Movie copyright" It defines the movie copy-right details. Typically indicates the owner of the copyright on the movie.

"Movie credits" It defines the movie credit details. Typically used for credits to those who provided resources for the movie.

QuickTime VR object movies

Object movies are slightly different in that they consist of a number of views of a static object taken from different points around it.

This allows the user to fly around the object (the effect is identical to, and often described as, rotating the object around any axis). As you turn the object, or tilt it up and down, QuickTime VR responds to your movements, displaying the correct image for that view onto the object.

The view resolution determines the dimensions of the movie, and the camera position is taken to indicate the front of the object that will form the movie subject.

***Note**: Object movie files will contain a large number of frames, and will therefore take a significant time to render.*

The middle **Parameters** column of the shading panel shows the following variables:

"Rendering method" Method used to render the frames of the movie. Wireframe and hidden-line are not currently supported.

"Pan min" The minimum horizontal pan position the object will be rotated to. Specified in degrees. Default is 0.

"Pan max" The maximum horizontal pan position the object will be rotated to. Specified in degrees. Default is 360.

"Pan initial" The initial horizontal pan position. Indicates the starting horizontal rotation for the object when the movie is first opened. Specified in degrees. Default is 180.

"Pan frames" The number of frames to generate between the minimum and maximum horizontal pan positions. Specified as an integer, defaults to 36 so that frames are rendered every 10 degrees.

"Tilt min" The minimum vertical tilt position the object will be rotated to. Specified in degrees. Default is -90.

"Tilt max" The maximum vertical tilt position that the object will be rotated to. Specified in degrees. Default is 90.

"Tilt initial" The initial vertical tilt position. Indicates the starting vertical rotation for the object when the movie is first opened. Specified in degrees. Default is 0.

"Tilt frames" The number of frames to generate between the minimum and maximum vertical tilt positions. Specified as an integer, defaults to 19 so that frames are rendered every 10 degrees.

"Movie info" It defines the movie information. Typically used for general information about the movie.

"Movie copyright" It defines the movie copy-right details. Typically indicates the owner of the copyright on the movie.

"Movie credits" It defines the movie credit details. Typically used for credits to those who provided resources for the movie.

5.14 Rendering the Scene

Rendering is defined as the process of generating a synthetic image of a scene given a complete description of its geometry and other characteristics.

The process of shaded image generation is divided into two distinct stages.

The first stage is known as visible surface processing, during which the renderer establishes, for every pixel of the image, what surface or surfaces are visible. When this information is available, the second stage of the rendering process evaluates the contribution made by each of the visible surfaces to the final shade of the displayed pixel.

This second step, known as shading, encompasses all of the calculations that must be performed in order to determine each pixel's intensity. These include accounting for every light source in the scene and calculating how much light falls on the visible surfaces, as well as determining the color of the visible surface.

solidThinking supports a number of choices for visibility processing in the form of alternative rendering methods.

Typically, each rendering method is designed to fulfill specific requirements and will provide a unique algorithmic solution for visibility processing. Depending upon the quality of image that is required and the features that it contains, ***solidThinking*** utilizes alternative methods of shading operation in order to maximize performance in various configurations.

Rendering methods can be summarized as follows:

Wireframe rendering

The simplest form of rendering. Vector output showing all the edges of the geometry with no consideration to whether a particular edge should be visible or obscured by other geometry.

Hidden line rendering

Vector output showing only those lines which would be visible from the current view point (i.e., lines hidden by other surfaces are removed).

Flat shaded rendering

Each surface is assumed to be opaque with a single, uniform color. Curved surfaces are approximated by a mesh of polygonal facets. In the shaded image, each facet is painted uniformly with an average color.

Gouraud shaded rendering

Each surface is assumed to be opaque with a single, uniform base color. Curved surfaces are approximated by a mesh of polygonal facets, and in the shaded image, colors are calculated at each vertex of each facet, then, these colors are blended across the facets to create smooth shading of curved surfaces. It is named after his French originator, Henry Gouraud.

Phong shaded rendering

Each surface is assumed to be opaque with a single, uniform base color. Curved surfaces are approximated by a mesh of polygonal facets, and in the shaded image, normal vectors at the vertices of each facet are interpolated and a shade is calculated independently for each pixel. This technique, known as Phong interpolation, enables smooth shading of curved surfaces and is able to render precise, realistic specular highlights. It was originated by Phong Bui-tuong.

Scanline preview rendering

Shaded rendering with full texturing features using the preview scan-line renderer. The geometry is sampled at a rate of one sample per rendered pixel, so that object silhouettes take on a jagged appearance. Transparency shading is not supported in this method.

Scanline full rendering

The model is rendered as a fully shaded and textured, anti-aliased image. The geometry is sampled at a sufficient rate to eliminate aliasing artifacts on object silhouettes.

Raytrace preview rendering

Shaded rendering with full texturing features using the preview ray tracer renderer. The geometry is sampled at a rate of one sample per rendered pixel, so that object silhouettes take on a jagged appearance.

Raytrace full rendering

The model is rendered as a fully shaded and textured, anti-aliased image using the full ray tracer renderer. The geometry is sampled at a sufficient rate to eliminate aliasing artifacts in regions of high intensity gradient.

Sketch rendering

The model is rendered using a technique which produces an image which is more like a pencil sketch, ink drawing, or painting.

The Render Options Panel

The **Render Options** panel (**Render ▸ Options…** command) allows you to specify rendering methods and resolution.

You can choose one of the following methods:

- **Wireframe**
- **Hidden line**
- **Flat**
- **Gouraud**
- **Phong**
- **Scanline preview**
- **Scanline full**
- **Raytrace preview**
- **Raytrace full**
- **Sketch rendering**

Renderings are automatically saved in the */solidThinking/ Images* folder with a progressive numbering "(Untitled)#*YYYY-MM-DD#hh-mm-ss*" where *YYYY* is for year, *MM* for month, *DD* for day, *hh* for hour, *mm* for minutes, *ss* for seconds, for example 2002-04-20#11-23-59).

The **Preset** pop-up menu lets you set the dimensions of the image using some commonly used settings.

The **Unit** field allows you to choose the reference unit (Pixel, Millimeter, Inch)nfor the below **Width** and **Height** fields. Width and Height can be swapped by pressing **Swap**.

When the chosen unit is Millimeter or Inch, the **Resolution** fields can be used to et the Pixel per Inch (dpi) or the Pixel per Millimeter values.

Once you have defined these options, you can start rendering your scenes.

Choose the **Render ▸ Render current view** command to generate the image at the fixed resolution.

The final image will be progressively displayed while it is computed. At the end of the process, the image titlebar will display the total rendering time.

The Renderings Browser

The **Renderings Browser** (**Render ▸ Browse Renderings…** command) shows you all the pictures you have rendered since you launched ***solidThinking***. Once you select a picture (just click it), you have the possibility to *save* it or *delete* it. The Renderings Browser also allows you to *delete all* the listed pictures.

When you click the **Save** button, a file requester that lets you save and name the rendered image appears.

You can set a maximum number of rendered pictures that will be displayed in the Rendering browser and stored in the */solidThinking/Images* folder in the **Renderings buffer** section of the **Preferences** panel (to open it, choose the **Info ▸ Preferences…** command).

Interreflections

A reflective object can reflect another reflective object. In real life, this creates a virtually infinite number of interreflections.

In ***solidThinking***, you can set the number of interreflections computed when you use the raytrace method.

To set the number of interreflections, open the **Shading** panel by using either the **Managers ▸ Shading** command.

Select the **Globals** tab: the first column on the left lists, among the others, the **Ray Traced** option. When you select it, a number of parameters will be displayed in the column beside.

The "**max reflection bounces**" parameter allows you to set number of interreflections either by moving the slider or by typing the value in the data field and pressing Enter.

Creating an Image Step by Step

Once you have modeled, refined, and positioned your 3D models, you are ready to produce a photorealistic image.

We report here steps that you could follow to produce images more efficiently.

1) **Set the camera position and the angle of view**.

2) **Assign simple materials**, for example materials whose components can be a "plain" color source shader, a reflection shader set to "matte", "plastic" or "metal" and no displacement or transparency.

3) **Position a few lights and set their intensities**. Most scenes only require a simple set of lights, an ambient light with a low intensity and a single distant light. The distant light is likely to be positioned at 45 degrees to the camera viewing line. As an alternative to the distant light, a spotlight may be used. By positioning the spot light carefully, the objects being rendered can be made to stand out from the background. At this stage shadows should *not* be turned on. A recommended lighting set up is an ambient light with an intensity of 0.2 and a distant light with an intensity of 1, positioned appropriately for the relative positions of the camera and the object.

4) **Use the Preview rendering for this stage**.

5) **Assign definitive materials**. Select a type of material applicable to each object. Plain materials are appropriate for most objects. If more complex materials are selected, like blue marble, it is likely that the parameters for the material will need to be adjusted for the particular object to which they are attached. The scale parameter of such shaders allows the size of the visual appearance of the shader to be adjusted to suit the size of the object to which the shader is attached.

The three simplest reflectance shaders are matte, plastic and metal. Matte is used for materials that do not have polished or glossy surface finishes, like paper and matte-painted walls. The plastic shader is applicable to materials that are glossy and have sharp highlights, like plastic objects, varnished wood, gloss-painted surfaces and glazed ceramic materials. The highlights on plastic materials will `take on' the color of the lights being reflected, normally a white color. The Phong reflectance shader is similar in effect to plastic. The metal shader may be used on metallic objects like machined mechanical parts. The material will have sharp highlights like plastic, but they will be a lighter shade of the base color, rather than the light's color. Most objects do not require materials that use ray-tracing reflection shaders, because few real-world materials have mirror-like reflections. Many materials can be simulated using the plastic and simple metal materials, which are faster than the ray-tracing materials. The chrome 2D reflectance shader, which simulates a simplified reflective surface, can be used as a simple alternative to the ray-tracing reflection shaders.

Only use shaders that use ray-tracing, i.e. mirror, glass, dielectric and conductor if necessary. The calculation time of these shaders may be reduced significantly if the mirror factor parameter of the shader is set to be 0. If this is done, then no ray-tracing will take place, increasing the performance and reducing the memory requirement.

There are two ways of rendering transparent objects, either through ray-tracing reflectance shaders or transparency shaders. Ray-tracing can simulate the complex refractions, or bending, of light as it passes through objects. For flat or thin transparent objects, like glass in windows where there is little refraction, transparency shaders are much more efficient than ray-tracing. The plain transparency shader gives materials a uniform transparency, and has as a parameter a color specifying the amount of transparency.

If the components of the color are set to 0.3, then the material will be two-thirds transparent. If the components are set to 0.9, then the material will be nearly opaque. By setting the color parameter to be a color, like red, the material will simulate colored glass.

Displacement shaders can simulate changes to the smoothness of a surface's finish. The rough displacement shader may be used for 'natural' bumpy surfaces like stone, cast metal and worn wood. Other displacement shaders like wrapped dimple can simulate manufactured surfaces, where a regular pattern appears across a surface.

6) **Adjust shadows**. The shadow resolution may need to be adjusted depending on the detail contained within the object and the overall size of the object. Some images require additional fill lights to lighten the shadows cast from the main light. A distant light positioned at an angle to the main light will accomplish this.

7) **Render the final image**. For the final image generation you must select an appropriate image resolution. The final image resolution and aspect ratio should be selected appropriately for the final output device. If the image is to be displayed on a computer screen, then the resolution should match the resolution of the display, for example 1024 by 768 pixels. If the image is to be output onto transparency film, by a film recorder, typical resolutions are 2048 by 1365 or 4096 by 2731 pixels. The quality of shadows is partly dependent on the resolution of the shadow map, and also the resolution of the image. For higher image resolutions the shadow map resolutions may also have to be changed appropriately. For the maximum image quality the Scanline full rendering mode should be used. If shader aliasing occurs then the Raytrace full rendering mode may be used, however this will significantly increase rendering times.

Rendering Hints

- **Rendering times are proportional to the resolution of the image**. An image with a resolution of 1000 by 1000 pixels will take approximately 4 times as long to render as one with a resolution of 500 by 500 pixels. To reduce the time taken for each iteration sub-images may be used to limit the size of the area rendered.

- **The time taken to render an image is affected by the rendering mode**. The flat rendering mode is the fastest, followed in order by Gouraud, Phong, Scanline preview, Scanline full, Raytrace preview and Raytrace full. The ray-tracing rendering modes are not normally required. Raytrace full may be used instead of Scanline full when shader aliasing occurs. The ray-tracing rendering modes use the ray-tracing technique to calculate surface visibility and are consequently slower than the scan-line based rendering modes. The reflection and refraction effects of ray-tracing are available in the Scanline preview and full rendering modes, combining the speed of scan-line visibility with the visual effects of ray-tracing.

- **Rendering time will increase with the number of lights**. Normally, most scenes require only two or three lights. The relative computation time required for each light type, in increasing order, is as follows: ambient, eye, distant, point and spot. Shadows from lights will increase the rendering time. Shadows are both computationally and memory intensive. Shadows from point lights are especially memory intensive. Turn shadows off during the iteration process. The normal resolution for shadows should be in the range of half the image resolution and the full image resolution.

- **Rendering times can be reduced and the quality of shadows increased** if objects that do not cast shadows onto other objects are excluded when a shadow is created for a light. For example, if an image is being created of a widget on a flat surface, then the flat surface will not cast shadows onto any other object and can therefore be excluded from the shadow creation process.

- **The number of materials used in a scene does not affect rendering time**.

- **The types of materials used in a scene may affect rendering time**. In fact, some materials are slower than others. Any materials that use ray-tracing reflectance shaders, like conductor, mirror, dielectric or glass, will be slower than the simpler reflectance shaders like matte, plastic, Phong and metal. Using the displacement shaders like rough and cast will increase rendering times. The image area covered by objects using the 'slower' materials will affect the rendering time. If an object is quite small in the image then attaching a 'slower' material to it will have a small effect on the rendering time. However if the object covers a large area of the image, then the effect on the rendering time will be greater.

- **Rendering time is affected by the complexity of the geometry being rendered**. Sub-images and selecting sub-sets of the geometry may be used to reduce the rendering time during the iteration process.

- **Using displacement shaders increases rendering times**. Rough and cast are especially computationally intensive.

- Plain colored color source shaders like plain are quicker than the more complex ones like simple wood and marble.

- The plain and graduated backgrounds are quicker than the clouds background.

Glossary

Aliasing
A reduction in image quality caused by representing images with discrete pixels. In rendering, aliasing often refers to an effect also known as the 'jaggies'. These occur when vectors or object edges are rendered with a jagged staircase effect, to which near horizontal or near vertical lines are particularly prone. Additional samples are taken to calculate a more accurate average.

B-Spline
A smooth curve defined by a series of control points.

Control point
A point in the world space that controls the shape of a curve or a surface. Control points of NURBS curves also have a weight factor which is initially set to 1.

Curve
The path of a point moving through space or a sequence of straight or curved lines. A curve is defined *non-rational* if all control points have the same weight factor. If they have different weight factors, the curve is called *rational*. A curve has only one dimension in space called U.

Displacement shader
One of the four component shaders used to define a material.
A displacement shader simulates the effects of small surface perturbations by modifying the surface normal vector which is applied to subsequent shading calculations. Also known as bump mapping.

DXF™
Drawing Exchange File. A file format widely used for transferring data between computer systems.

Field of View
A means of specifying which part of a scene is visible from the view point using the camera model of image control.

Gouraud shading
In the Gouraud shading technique, the color of pixels in the interior of a polygon is calculated by interpolating the values of the pixel intensities at the vertices. This produces an illusion of smoothness at relatively little computational cost. Proposed by Henri Gouraud.

Hidden-line rendering method
A vector rendering in which those edges or parts of edges that are obscured from the view are not rendered.

IGES™
Initial Graphics Exchange Specification. A file format widely used for transferring data between computer systems.

Interpolation
Finding a curve or surface that satisfies some user-defined constraints exactly. AN example of constraint is the requirement of passing through a number of given points.

Light source shader
A shader function that describes how much illumination from a light source falls onto a surface. A light shader may take one of several light types, such as point, distant or ambient.

Local coordinate system
A coordinate system that is applicable to a single object having its own center as origin. Also known as the *Object coordinate system*.

Material
The information which determines how an object will appear when rendered. In shaded rendering modes the appearance is derived from four shaders that determine the color, transparency, illumination model and surface displacement of the object.

NURBS (Non-uniform Rational B-Spline)
A curve definition method that uses specific B-Splines with weighted control points. NURBS offer the greatest flexibility and precision.

Pixel
Short for 'picture element'. Images are composed of arrays of pixels. Each pixel has a value which depends on the image format or display capability. Each pixel in a monochrome image has only an 'on' or 'off' state. Pixel values for color images are represented by RGB (red, green, blue) triples.

Phong shading
This technique enables smooth shading of curved surfaces and is able to render precise and realistic specular highlights by using three main characteristics: diffusion, specularity, and shininess. It was originated by Phong Bui-tuong.

Radiosity
A lighting simulation technique which works by calculating the transfer of radiation (light energy) from one surface to another. By performing this calculation repeatedly, for every pair of surfaces in the scene, the distribution of the light energy through the scene is modeled. This technique, unlike other global illumination models, such as ray tracing, accurately models the effect of diffuse inter-reflections in the scene—however it does not model specular reflections very well, and should so be seen as complementing means such as ray tracing, rather than replacing them.

Ray Tracing
A rendering technique in which the colors are computed by tracing imaginary rays of light through an environment. Ray tracing can be used to produce photo-realistic rendering effects, such as mirror reflections and refractions.

Reflectance shader
One of the four component shaders that contribute towards a material. A reflectance shader models the interaction of light with a surface and, in particular, how much light is reflected towards the viewer. A reflectance shader can be thought of as defining the ‘finish’ of surface.

Refraction
The change in direction of light as it crosses the boundary between two transmitting media.

Rendering
The complete process of producing an image from a model. This comprises two major stages: visibility determination and shading.

Resolution
The number of pixels in the horizontal and vertical dimensions of a display.

Scan-line rendering
Rendering which employs a scan-line algorithm. Scan-line algorithms operate by scanning each row of pixels in sequence to determine the visible surface and pixel value. Scan-line rendering can produce images more efficiently than ray tracing methods.

Shader
A shader or shader function can loosely be defined as a generic C-code function that calculates some aspect of the rendering environment such as a pixel's background color or the effect of a surface's transparency on the color of a pixel. Each shader has the notion of a named list of arguments or parameters.
Shaders are categorized into several different shader Classes.

Shadow
That part of an object or region of space not directly illuminated by a particular light source. The shaded rendering methods, flat, gouraud and phong do not support shadows.

Shadow map
A depth-buffered image computed from the view point of a light source to support the rendering of shadows on object surfaces. Shadow maps can be computed once by an application and reused for any set of viewing parameters until the position of a light or an object's geometry changes. As for other rendered images, shadow maps can be computed to an application specified image resolution.

Surface
A parametric shape having U and V dimensions in space. Any point on a surface can be identified by specifying U and V values.

Tessellate
To divide a curve or a surface into geometric forms for simplified processing and rendering. NURBS curves and surfaces are curved in a continuous manner, while computer display better deals with straight lines and flat polygons. For this reason, NURBS curves and surfaces are tessellated into polylines and polygons at redrawing time.

Texture
A material that renders an object as if it were carved out of a single piece of one material. Solid textures calculate shading values for points on the surface of objects as a function of their *three-dimensional coordinate system*. This is in contrast to the use of parametric coordinates by wrapped textures, which operate within a *two-dimensional coordinate system* known as *texture space*.

Texture mapping
A technique for rendering fine surface details on an object without explicitly modeling the geometry. In texture mapping, a two-dimensional image, known as a texture map is projected onto the surface of an object. A texture map may be either a procedurally defined function or an image, typically scanned or pre-computed.

Texture space
A two dimensional coordinate system used to map a wrapped texture onto the surface of a primitive.

Texture Space shader
Texture space shaders differ from other classes of shaders in that they do not affect the final intensity of a pixel directly. Rather, they have an indirect effect by defining the texture space that may be used by other shaders.

Transformations
Operations that move, re-orient, or resize items in a 3D scene.

Transparency shader A transparency shader defines the transparency of a surface and thus how much light can pass through it. A transparency shader deals with two separate phenomena: filters and stencils. Respectively, these define the color and presence of light passing through the surface.

Vertex
The location at which faces or edges of a polygon intersect.

Wireframe
An image made up of the edges of objects and drawn as lines. Wireframe models can be drawn much more quickly than surface models. The disadvantage is that in a complex 3D model, it may be difficult for a viewer to make visual sense of the drawing.

World coordinate system
A mathematical system of axes in which any point in the 3D scene is identifiable by one coordinate in the X direction, one in the Y direction, and one in the Z direction in relation to a world origin. Also known as the *XYZ coordinate system* or *Cartesian coordinate system*.

Shortcuts for Macintosh

General

Apple + Mouse Button
Scroll of Modeling, Materials and Models toolbar

Apple + **Enter**
Skip all console prompts

Esc
Abort a modeling tool previously activated

Apple + Spacebar
Recall last modeling tool activated

Ctrl
Hold down while picking for multiple selections of objects/points

Shift
Hold down while picking to choose a range of points from the currently selected to the clicked point

Alt
Hold down while picking to select an entity (and not the whole object)
Hold down while picking to force selection of another object when in Edit mode

Transformations

E
Elevation mode

T
Translate

R
Rotate

S
Scale

X
Constrain on the X axis

Y
Constrain on the Y axis

Z
Constrain on the Z axis

File

Apple + **N**
New

Apple + **O**
Open

Apple + **M**
Merge

Apple + **S**
Save

F12
Save as

Shift + **F12**
Save selected

Apple + **F12**
Save as model

Apple + **Q**
Exit

Edit

Apple + **Z**
Undo

Apple + **A**
Redo

Apple + **X**
Cut

Apple + **C**
Copy

Apple + **V**
Paste

Alt + **Backspace**
Delete

C
Collapse Construction tree

U
Uncombine entities

Apple + Alt + Backspace
Prune

Spacebar
Toggle *Object mode / Edit mode*

Alt + Spacebar
Toggle *Edit object's points / Edit Modeling Tools parameters*

Apple + Shift + G
Grid setup

Selection

Apple + G
Group

Apple + U
Ungroup

View

V
Enlarge the active view to fill all the space inside the main window

Apple + Mouse Button
Click and drag to pan while holding down the **Apple** key in Orthographic views
Click and drag to orbit while holding down the **Apple** key in Perspective view

Apple + Ctrl +Mouse Button
Click and drag while holding down **Apple** and **Ctrl** key to pan in any view

Apple + Shift + Mouse Button
Click and drag while holding down **Apple** and **Shift** key to dolly in any view

Apple + Alt + Mouse Button
Click and drag while holding down **Apple** and **Alt** key to perform area zoom in any view

Apple + L
Layouts

Apple + Shift + D
Detail

Apple + Shift + C
Curvature Display

Managers

Apple + 1
Browser

Apple + 2
Modeling Tool

Apple + 3
Shading

Alt + 1
Application toolbar

Alt + 2
Snaps toolbar

Alt + 3
Modeling toolbar

Alt + 4
Materials toolbar

Alt + 5
Models toolbar

Render

Apple + R
Render current view

Alt + Shift + R
Render selected objects

Apple + Shift + R
Options

Apple + Alt + R
Browse renderings

Shortcuts for Windows

General

Ctrl + **Enter**
Skip all console prompts

Esc
Abort a modeling tool previously activated

Ins
Recall last modeling tool activated - this tool can also be performed using Ctrl + Spacebar

Ctrl
Hold down while picking for multiple selections of objects/points

Shift
Hold down while picking to choose a range of points from the currently selected to the clicked point

Alt
Hold down while picking to select an entity (and not the whole object)
Hold down while picking to force selection of another object when in Edit mode

Transformations

E
Elevation mode

T
Translate

R
Rotate

S
Scale

X
Constrain on the X axis

Y
Constrain on the Y axis

Z
Constrain on the Z axis

File

Ctrl + **N**
New

Ctrl + **O**
Open

Ctrl + M
Merge

Ctrl + S
Save

F12
Save as

Shift + F12
Save selected

Ctrl + F12
Save as model

Alt + F4
Exit

Edit

Ctrl + Z
Undo

Ctrl + A
Redo

Ctrl + X
Cut

Ctrl + C
Copy

Ctrl + V
Paste

Del
Delete

C
Collapse Construction tree

U
Uncombine entities

Ctrl + Del
Prune

Spacebar
Toggle *Object mode* / *Edit mode*

Alt + Spacebar
Toggle *Edit object's points* / *Edit Modeling Tools parameters*

Ctrl + Shift + G
Grid setup

Selection

Ctrl + G
Group

Ctrl + U
Ungroup

View

V
Enlarge the active view to fill all the space inside the main window

Right Mouse Button
Click and drag to pan in Orthographic views
Click and drag to orbit in Perspective view

Ctrl + Right Mouse Button
Click and drag while holding down the **Ctrl** key to pan in any view

Shift + Right Mouse Button
Click and drag while holding down the **Shift** key to dolly in any view

Alt + Right Mouse Button
Click and drag while holding down the **Alt** key to perform area zoom in any view

Ctrl + L
Layouts

Ctrl + Shift + D
Detail

Ctrl + Shift + C
Curvature Display

Managers

Ctrl + 1
Browser

Ctrl + 2
Modeling Tool

Ctrl + 3
Shading

Alt + 1
Application toolbar

Alt + 2
Snaps toolbar

Alt + 3
Modeling toolbar

Alt + 4
Materials toolbar

Alt + 5
Models toolbar

Render

Ctrl + R
Render current view

Alt + Shift + R
Render selected objects

Ctrl + Shift + R
Options

Ctrl + Alt + R
Browse renderings